NOVA IMS
Information Management School

MAA

Mestrado em Métodos Analíticos Avançados
Master Program in Advanced Analytics

Simulating Traffic Patterns of Cars as a Result of Closing a Specific Road

Alex Anthony Panchot

Dissertation presented as partial requirement for obtaining the Master's degree in Advanced Analytics

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

NOVA Information Management School

Instituto Superior de Estatística e Gestão de Informação

Universidade Nova de Lisboa

SIMULATING TRAFFIC PATTERNS OF CARS AS A RESULT OF CLOSING A SPECIFIC ROAD

by

Alex Anthony Panchot

Dissertation presented as partial requirement for obtaining the Master's degree in Advanced Analytics

Advisor: Prof Doutor Flávio Luis Portas Pinheiro

Co Advisor*:* Prof Doutor Miguel de Castro Neto

Acknowledgments

I would like to thank my supervisor Flávio Pinheiro for guiding me through the dissertation process. Additionally, I would like to thank Nuno Alpalhão for helping me with and Miguel de Castro Neto for providing me with the NOS data.

I would like to thank OpenStreetMap and OpenStreetMapX for providing the map and map parser respectively.

Abstract

While some analytical solutions approach a problem with existing data, agent-based simulations give the ability to approach the problem with data from the ”future”. The problem tackled in this work is the prediction of specific traffic patterns (namely traffic jams) for the city of Lisbon. Specifically, the question was to see if we can model the change in traffic after a road closure, such as for construction. This question is an interesting one as it explores a logistical problem that every driver and city planner has faced. Additionally while on the surface it seems focused on just traffic and city streets, general problems of path finding and global optimization are also explored. While there exist other software to simulate street traffic, none are very simple and require every location to be manually created for that particular software. The proposed solution is twofold. The first is to design a reproducible method for creating the agents from either existing data or from a set of rules that can be used to create agents where there is no existing data. The second is to design a simulation algorithm that can run off of an existing map, in this case open-source OpenStreetMap, such that the map can instantly be swapped with any other with no changes to the code required. By designing a simulation like this, the ability to simultaneously compare multiple modes of transportation can also be realized. The final results show that this project is successful as a way of seeing at least general trends as a result of a change in the road network (a road closure). While more work is needed to add multiple modes of transportation, the ability for the simulation to run on a different road network allows it to be easily extended in this area as well as other graph networks or flow simulations. While not perfect, simulations can be a significantly cheaper and safer option for analyzing changes to a network than actually making that change and analyzing what happened after the fact.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

As cities grow larger and more interconnected, transportation and its problems become more important to think about and address. One of the major issues surrounding transportation is congestion in major cities (Rodrigue, 2020). As traffic congestion slows the speed at which people and goods move, economic growth can become hampered (Rodrigue, 2020). Hence, it is of major importance for urban planners and administrators to understand and predict mobility patterns to support a more sustainable and efficient city development, in which includes the ability to anticipate and overcome the occurrence of rare events.

A popular approach to study urban mobility is to rely on computer simulations. These can include multiple modes of transportation in order to generate realistic behaviors. Although fictitious, simulations give us a more thorough analysis of different scenarios rather than just relying on those available in existing and past data. Furthermore, simulations allow the modeler to make changes (sometimes in real-time) to the system and develop a more comprehensive understanding of the possible outcomes that would be otherwise expensive and time-consuming to perform in real life.

To simulate mobility patterns at the city-scale there are different methods available such as Discrete Event Simulation, System Dynamics, and Agent Based Simulation (Maidstone, 2012). However, approaches based on multi-agent systems appear as superior choices (Bazghandi, 2012), as they provide the ability of the modeler to microscopically define the behavior of each agent (part of the system) while providing insights of the behavior of the whole system.

Hence, the main goal and motivation of this thesis project was to develop a computer-based multi-agent based model that would be capable of realistically simulating the mobility patterns of citizens through the city of Lisbon by way of various modes of transportation. These modes were to be bicycles, public transportation, and personal cars. However, given the complexity of the task at hand, it was decided to focus, as a first step, on a model that would cover car traffic and then, by ensuring the modularity of the model, possibly extend to other modes in the future. Therefore, the two main questions that we seek to answer in this work are:

1. Can an agent-based model based on simple premises capture the complex behaviors in urban traffic mobility?
2. Can we use such a model to predict the impact that an external event, for instance from the closure of a major road, would have on traffic flows?

Despite the model being designed primarily for cars (the other modes were not built in), not too much additional work should be needed to integrate the other modes as the model was designed to ensure it is modular, generic, and flexible.

Chapter 2

Literature Review

Congestion is caused by two main factors. First, too many cars on the road slows down traffic as the cars must avoid hitting each other; secondly, those cars spend additional time looking for a place to park. In larger cities it is possible that up to 10% of active drivers are no longer commuting, but simply looking for a place to park (Rodrigue, 2020). Because of the added delays, drivers are forced to spend more time on the road which increases traffic even more. Additionally, more traffic means more pollution and more wear and tear on the roadways causing them to breakdown and need more repair (which itself causes traffic). High levels of traffic also tend to dissuade people from using other modes of transportation, such as walking or biking (Rodrigue, 2020).

As vehicle congestion exists all over the world, many cities have taken different steps to try and solve these problems (Rodrigue, 2020). Some of the steps are designed as carrots while others are more like sticks. Some cities, such as Washington D.C, have introduced bicycle programs which have lowered traffic congestion by 2% to 3% in a neighborhood (Hamilton & Wichman, 2015). Others, like Houston, have increased public transportation access and added lines and stations (Sisson, 2017). Conversely, others, like London, have increased/created congestion charges on cars (especially in certain areas and for certain times of day) (Badstuber, 2019). While every city takes a slightly different approach, no method is perfect as action creates additional traffic consequences.

As traffic is a perennial problem, many groups around the world have attempted to model the behavior of traffic such that they can suggest solutions to fixing it (Pursula, 1999). Depending on the type of traffic analyzed, two main groups of models may be used. The first are mathematical models such as differential equations, such as ordinary or partial differential equations, which are useful for a continuous time dimension . For when the time dimension is discrete, various types

of simulation, such as Discrete Event, System Dynamics, or Agent Based, can be used (Maidstone, 2012). These simulations can be very different from each other as the only real similarity is that there exists a discrete time step in which actions can be taken. These actions happen because of previous time steps but not of the current one, as that falls into the realm of continuous.

2.1 Agent Based Simulations

Simulations are an excellent choice, and sometimes the only one, for realistically simulating complex and dynamical systems. Moreover, when it comes to simulations, there are three main types of simulation approaches that can be developed at three different scales. These differences will be discussed here so that it is clear why agent-based modeling is the most appropriate for this type of simulation.

The first simulation approach, Discrete Event Simulation (DES), is the most widely used type of simulation (Maidstone, 2012). This is because DES models explicit and discrete events. The example given by Maidstone is one of a hospital emergency room where each patient enters and goes through specific steps before they are released. An important distinction to make is that each patient does not directly impact the other ones. Under normal circumstances the care of one patient has no impact on the care for a different one.

The second type, System Dynamics (SD), differs from DES in that it simulates the system rather than each individual component. This means it looks at the flows and capacities of each component of a system and, because of such constraints, the behavior of the system can be predicted by the shape of its components. The example of SD Maidstone gives is one with an entire hospital. Instead of focusing on individual steps or patients, SD defines flows between each of the departments. This could be calculated by taking the average number of patients that move between each department in a day. This would allow a hospital to gauge whether or not they have sufficient capacity in each of their departments.

The third type, Agent Based Simulations (ABS), is closely aligned with DES models but their characteristics differ quite significantly. The premise of agent-based models is to define a group of individuals and the interactions that can take place between them. As the agents update their own behavior in real time from the inputs of their interactions with others, the system can produce stochastic and unpredictable results. Giving agents the ability to learn, either through simple means—such as by imitation or reinforcement—or more complex ones—such as the use of AI based systems—makes the system even more susceptible to random changes and unpredictable

results. Bonabeau (Bonabeau, 2002) defines ABS as more of a mindset than a specific technology. This is because unlike differential modeling that has specific equations to follow, ABS is very flexible and easy to implement. For example, while a traffic flow model with differential equations would require exact values for the density of cars, ABS allows for those values to be calculated by the simulation itself, rather than as a prerequisite. In the same vein, ABS is more natural to describe and build. If we want to model cars, we only need to model what each car does and how it interacts with its nearest neighbors. Further ranged interactions are taken care of by the system itself.

Some of the areas of application include agent mobility markets, and diffusion of information (Bonabeau, 2002). In the case of agent mobility, they have been used to model individuals' mobility through various scenarios such as, for instance, traffic, evacuations, and customer flow through a shopping center (Bonabeau, 2002). Additionally, simulations can be run to analyze supply chains, customer behaviors, and human health (such as the spread of epidemics and the study of cells and molecules) ("4 Agent Based Modeling Examples", 2019).

Traffic simulations can be very complex as they describe how and why an individual makes a journey between two locations. When an agent makes a journey, they must examine both their past experiences (when to leave and/or choose the best mode of transportation) as well as current situations (detours or possibly changing the mode of transportation). As traffic is a result of individual behaviors, every change in action by a single agent has a change on the system itself. These simulations can also be modified to examine how internal (change in personal behavior) or external forces (road closures or limiting cars from certain areas) affect the agents and the resulting information gathered from the agents.

The usefulness of ABS is best seen when the interactions between the individual entities are complex and/or discrete. The complexity can arise when the individuals and/or the topology of the interactions are heterogeneous. This is the case is nearly every human-to-human interaction as each person exhibits different behaviors and reactions. Humans tend to also be more individualistic and are capable of learning in real time which means that even if two individual agents begin with the same behavior, that behavior may be completely different by the end of a simulation.

While ABS is very powerful, some critical issues remain with its implementation. As the agents must be defined with specific behaviors, ABS models might lack generalization without significant changes to fit a specific problem/context. Since simulations can be unstable (i.e., dynamically chaotic), getting the initial conditions right is a challenge. Besides these difficulties, the simulation of many agents can be very computationally intensive. While computer resources have grown immensely in recent years, so to have the complexities of the models being simulated (Bonabeau, 2002).

When designing an ABS simulation, an important consideration is to what level of detail it will run at or the scale at which the problem should be correctly analyzed. Simulations at the macro-scale look at the whole system and are more similar to System Dynamics. For example, a macro-scale simulation would consider cars on a road as equivalent to liquid in a pipe. Simulations at the microscopic scale are much more realistic as they consider each car and can move each car with a speed that matches the one in front of it. However, as this type of detail can be computationally very expensive, meso-simulations (in between full macro or micro) can be more appropriate. One example of this kind is to treat each road as a queue and only worry about interactions at intersections of roads rather than along the road itself.

2.2 Agent Creation

The core challenges of developing an ABS model are to correctly define the agents, their decision heuristics, define how agents' heuristics are revised, and the agents' objective.

As the creation of the agents is central to any simulation's success, the first part of chapter 4 is focused on this topic. Additionally, since many of these simulations and algorithms are custom built for a specific region or use data that is specific to a region, we cannot easily produce realistic agents from an already existing algorithm made for a different region to use in our region of interest.

2.3 Human Mobility Simulations

Most simulations focus on a specific type of transportation and model how users may use that particular mode. Typically, these systems are modeled individually such that they do not consider other modes of transport. Focusing on only one mode of transport has the benefit of being faster to setup, implement, and analyze; but it also has the downside of assuming that all modes of transport are independent.

To get an overview of the different modes, we can look at some past works for each mode of transportation.

Cars: simulations of car movements are perhaps the simplest of all transportation modes. This is because cars can only travel on roads and take up a defined space on the roadway. Typically, car simulations only become complex with micro-interactions as described previously. Drivers can be modified so some are more aggressive or experienced, while cars can be modified such that the acceleration or size is different (Karima et al., 2012) The roadway itself can also be changed to reflect real world sizes or conditions;

Pedestrians: Pedestrian simulations, at first glance, are perhaps more closely related to macro-simulations. As pedestrians do not need to follow specific rules or roads, they typically will move along the physically shortest route. But they may diverge from the shortest route as it becomes filled with other people or an object is blocking the way, much like how gas particles move. This is because humans can turn and accelerate relatively much more quickly than cars. These tiny changes in space can be mathematically calculated with tiny changes in time and used to trace the movement of a whole group (Kimura et al., 2018);

Public Transportation: Public Transportation can be separated into two different categories, open and closed. Closed systems are like trains which should (theoretically) always be on time as they only depend on internal factors. Open systems, like buses, depend on external factors such as other drivers. This makes open systems much more difficult but also interesting to model;

Taxi/Ride-Sharing: Taxis and other ride-sharing schemes offer the benefits of a personal car (door to door service) while also keeping the benefits of public transportation (no need to maintain or park vehicle). This comes at a considerable per use cost, if one is to use this mode of transportation regularly. Therefore, this mode is typically reduced to specific use case scenarios. In order to address the cost associated with taxis, a paper (Martinez, 2017) looks at the efficacy of adding ride-sharing as well smaller buses that must be pre-booked before use. Both of these options are designed to combine the convenience of a traditional taxi with the space efficiency (and cost reduction per user) of a larger shared vehicle, like a bus;

Bicycles: Bicycles are an interesting mode of transportation as they encompass some of the features of all of the other modes. Bicycles are faster than walking (except for hills or poor terrain), while also being small enough to avoid being stuck in traffic or needing to find a place to park. Besides a person owning their own bicycle (which is equivalent to walking really fast), there are two types of shared, rental bicycle options available. The first is free-floating bicycles that can be picked-up and dropped-off at any location. These are somewhat similar to taxis as they are point to point transportation and do not require the operator to find parking. The second option is bicycles that are docked in stations. This is somewhat similar to public transport as there are defined stations, but crucially, the availability of bicycles and parking is dependent upon other users and the company.

The evolution of shared bicycles can also be clustered into several phases (Soriguera et al., 2018). At first, bicycles were completely free to use, but overtime evolved to require deposits. These systems further evolved to more personal identification in order to limit damage to the system. Today's systems are connected to the internet to allow for much better tracking of each of the bicycles in the system. This has also allowed for the introduction of the dockless bicycle.

However, docked bicycles are still better than dockless for the operators. As the bicycles are docked, they can charge and as there are defined stations, people cannot leave the bicycles wherever they want to. As a result of the limited number of stations, there will be a natural emptying of certain stations while other ones become too full. The solutions to this are manual balancing as well as designing a more optimal system with more stations in certain areas.

One simulation by Soriguera (Soriguera et al., 2018) studies the manual positioning of the bicycles between stations by company trucks. In this simulation, agents are using the system and walk to a station and use it, if a bicycle is available at that station. As there is a maximum distance that an agent will walk to a station, a station with available bicycles must be within walking distance. Electric bicycles also have additional demands such as they have a maximum range and require charging time between uses. The simulation then focuses on a algorithm of the best way to transfer bicycles between stations by truck.

Beyond the bicycle system itself, riders typically have many more considerations when it comes to the route than someone who drives or walks. For example, a simulation by Ziemke (Ziemke et al., 2017) uses the slope, pavement type, and riders' interactions with other vehicles (through shared or separate bicycle lanes) as variables. After including these and other variables the route a rider takes changes. Riders are willing to increase their trip time in order to avoid poor quality or crowded roads.

From these two papers, it is clear that bicycles are very complex to model as a rider's desire and ability to ride depends on the availability of the bicycles as well as the quality of the roads along their route.

Chapter 3

Methods

In this section we revise some fundamental theoretical mathematical methods necessary for the better understanding of the work that was conducted in this thesis project. While other mathematical tools are required, the two methods presented here are not as common and therefore deserve a more thorough explanation.

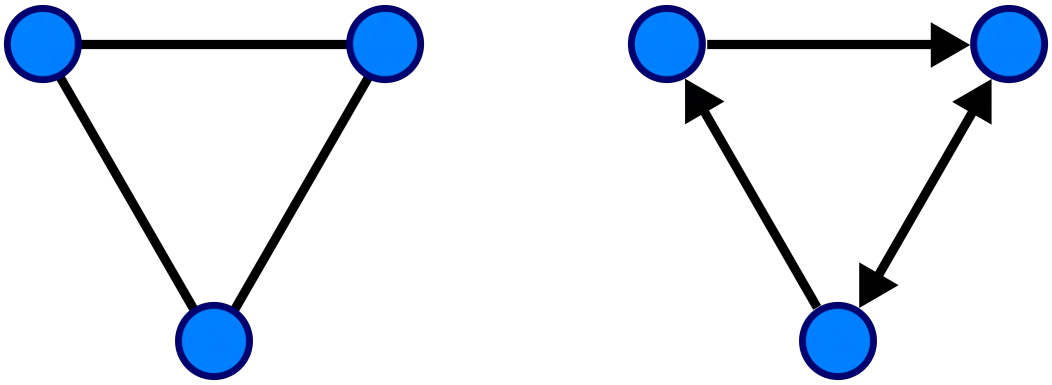
3.1 Graphs

The discrete mathematical graphs described in this section are the basis for many systems that need to store relationships between discrete objects. For example, these systems can be non-tangible social networks, the physical layout of streets in a city , or even the flow of fluids through various pipes and other connectors (Easley & Kleinberg, 2010).

A graph can be created by having different objects (or also known as vertices) and the connections between them (or also known as edges). These connections can either be un-directed or directed. An un-directed graph is one in which the relationships are mutual between both objects. For example, a handshake occurs between two people in a mutual and identical way. Directed graphs are the opposite and can occur whenever a connection is not identical, such as in a parent/child relationship.

Figure 3.1 shows the way by which a graph can be visually represented. Un-directed graphs show solid lines to indicate equality, whereas directed graphs show arrows to show some sort of un-equal connection (and the direction of the connection, from source to destination). Directed

graphs may also have some connections equal (with the connection represented by two arrows, one in each direction) such that the idea of directed and un-directed is not mutually exclusive.



Undirected Graph Directed Graph

Figure 3.1: Example Graphs

While the basic idea of a graph is simple, the attributes assigned to the graph is what make graphs quite powerful. In the context of a city street, one may find that each intersection can be a vertex and each road, connecting two intersections, an edge connecting the two vertices. In order to move between two points in a city, one may want to find the shortest route available. This can be achieved by creating a ”weighted graph”. This kind of graph means that the edges do not just represent that a connection simply exists, but rather the connection has some sort of weight attached to it. In the case of a city road, the length of the road is a simple and typical weight one may use. With these weights, route-finding algorithms can determine which roads to take based on just the length of each road (in a simple case). Edges may also have multiple weights that could be used differently in different contexts. For example, a bicyclist may find the slope of a road very important, while a car driver may not. So a bicyclist may choose to take a longer route horizontally if it means that they can avoid a large hill.

3.2 The A^* Algorithm

The A^* algorithm (Hart et al., 1968) is a route-finding algorithm that aims to find the lowest cost to travel between a set source vertex and destination vertex. The definition of cost is important to define before running this algorithm. As any route can be broken into several edges and vertices, the optimal route (chosen by A^*) will be found by trying to minimize the cost of traveling along the edges between the source and destination. If we define the weight of each edge (road) to be its physical length, A^* will return the shortest route in distance. This happens as it is minimizing

the cost of that route, which we in this case is the sum of the weights of the roads traveled. But we also may wish to define the edge weight to be a different value such as the time taken to cross each road. In this case, the result is still the sum of the individual weights, but it represents the shortest route in time not necessarily distance. We may also wish to define the weight to represent a ”good” thing such as road quality. So a larger weight may mean that particular edge is more, not less, desirable. In this case, either the cost function needs to change to maximize the weights, or the weights need to be inverted. Inversion is simpler as the same algorithm code can be reused, but it loses some of the original meaning of the (larger is better) data.

A^* searches by systematically exploring specific vertices until it finds the destination vertex. The algorithm works in the following general fashion:

1. For the vertex that has just been visited, compute and assign the vertex the weight of the cost function $g(n)$.
   (a) The function $g(n)$ is given by $g(n) = f(n) + h(n)$.
   (b) The function $f(n)$ is the current sum of the weights of all edges from the current vertex back to the source destination (or how far the current vertex is from the source).
   (c) $h(n)$ is a heuristic function that computes the distance from the current vertex to the destination vertex. Crucially, this distance is not a sum of weights of any edges (as we could not previously have this knowledge). This distance can be something as simple as just the straight line distance (in meters) between the current and destination vertex.
2. For all of the current neighbors of the current vertex repeat step 1 for each of them.
3. Look at all vertices that have been given a weight, but have not yet been chosen and pick the vertex with the lowest cost.
4. If the neighbor picked is the destination, stop; else repeat steps 2-4.

Algorithm 1: A^*: Finds shortest path between two vertices on a weighted graph

Input: A weighted graph; A start location; And an end location
Output: A list of locations that form the shortest path between the starting and ending locations

```
1  current_location ← start_location
2  neighbors ← [ ]
3  while current_location does not equal end_location do
4      n ← current_number_of_neighbors
5      for i ← 1 to n do
6          if neighbors does not contain neighbor_i then
7              g_value ← f(neighbor_i) + h(neighbor_i)
8              neighbor_i.value ← g_value
9              append neighbor_i.value to neighbors
10     past_location ← current_location
11     current_location ← minimum(neighbors)
12     remove current_location from neighbors
13     current_location_came_from ← past_location
14 shortest_path ← [ ]
15 while current_location does not equal start_location do
16     append current_location to shortest_path
17     current_location ← current_location_came_from
18 return shortest_path
```

Figure 3.2: Pseudocode of the A^* Algorithm

The algorithm works as the additional $h(n)$ function will always direct it to pick the vertices that are going in the correct direction and will ignore vertices that are further away, even if they have a smaller sum of edge weights ($f(n)$) back to the source vertex.

The A^* was selected for this work as it is one of the best on average performers (it searches the fewest number of vertices to find the best route) in general as well as requiring very little, if any, preprocessing of the graph to begin finding a route (and it was one of only a few already implemented in code algorithms easily available). However, other techniques can be used to significantly (several orders of magnitude) improve performance. These can require various levels of preprocessing overhead, but in the case of a static graph that is used many times (such as the ones used here or in any map models) the initial overhead is probably worth the performance increase.

A^* is considered complete and guaranteed to terminate. This means that it will always find a route and will terminate if one does not exist. It will not get stuck in a loop or attempt a deep search if one does not exist. It is also considered to be admissible in the sense that the route that it finds will be the optimal route. However, this is only true when the heuristic used is consistent. In this context, the cost of moving from any vertex to a destination should be less than or equal to the sum of the cost of moving to any other vertex and the cost of that vertex to the destination. In this sense, it is equivalent to the triangle inequality. However, one may choose a heuristic that is not consistent and thus will result in a sub-optimal route being found.

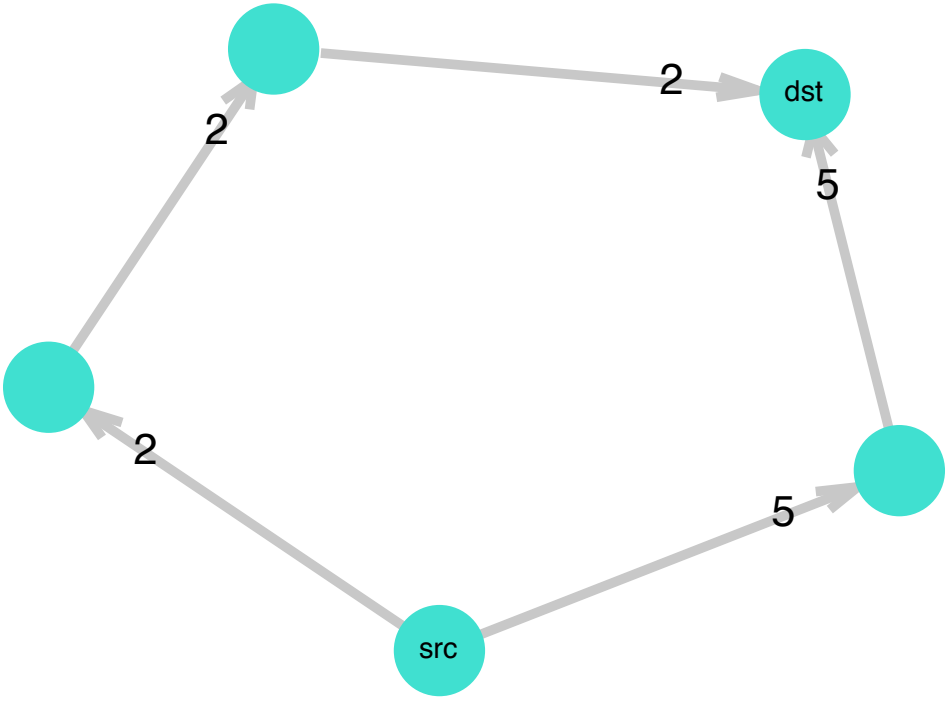


Figure 3.3: A^* Algorithm Example

In Figure 3.3, a simple visual example of A^* is shown. The source vertex (src) and destination vertex (dst) have two routes between them. The optimal route is the route that has two intermediate vertices rather than the route with only one vertex. This is because the route with two vertices has a total edge weight of 6 $(2 + 2 + 2)$ and the route with one has an edge weight of 10 $(5 + 5)$. Since the vertices themselves do not have any associated weight, the number of vertices, or edges, is irrelevant. Additionally, we should note that looking at all of the total weights in advance is not how the algorithm actually works. Doing this would be tantamount to a deep search.

Chapter 4

Agents

Before we can examine how the simulation model works, we will look at the agents that will be used. One of the ways to generate agents is by using real data from various sources. Some demographic aspects such as car ownership can be derived from census data, but this can be rendered useless if the data is too high grained. Additionally, some data such as origin and destinations for the agents can only be collected by travel surveys. Unfortunately, these surveys usually do not reach more than 2% of the population (Viegas, 2010). This low percentage is problematic as it could have biases towards the population that fills out the survey. As using these surveys is not ideal (and running a new one would be prohibitively expensive and take too much time), we need to look for another way to generate agents.

As the simulation requires a map as an input, we can use that to generate the agents. A good choice for the source of the map is OpenStreetMap (OSM) because OSM is open source and any map of the world can easily be extracted and be used as an input to the simulation without any extra work from the user. Besides the street data, the map also provides lots of data on what type of buildings are next to these roads. What we are interested in is the so-called "land use" of each location. The land use is what that particular piece of land is being used for, this is also known as zoning. For example, we may have a residential area next to a commercial one next to a school and so on. While some types of area only have one tag, such as residential or school or university, there are also at least six different tags for types of green spaces: forest, wood, park, common and meadow, heathland, and nature reserve. These six are also only for natural green spaces. Man-made spaces such as recreational parks or fields are different tags.

What we can do with this information is draw from it the origins/destinations. For example, we would probably want each agent to start/end their day in a residential area, so we can force the first origin and last destination to be in a residential area.

Figure 4.1: OSM Land Use (OpenStreetMap contributors, 2017)

Looking at Figure 4.1, we can see on the right side is a key which lists the various land use colors. Underneath the buildings (which are the tan/gray polygons) we can see some of the land use colors such as pink for retail or commercial areas or gray for residential areas or green for green spaces. The basic steps for finding these land use tags are as follows:

The OSM map is somewhat confusing, but each connected list of points is stored as a ”way”. This includes both 1D lines (roads) as well as 2D polygons (various zones, including land use!). Each ”way” has a set of tags, with ”amenity”, ”landuse”, and ”leisure” identifying land use zones (”amenity” is for more specific places such as ”bank” or ”theatre”, while ”leisure” focuses more on recreational amenities such as a parks, gardens, arts centers, golf courses, etc. ”Landuse” is more generic, like ”residential” or ”commercial”). If the ”way” has either tag, they we can also see what specific place that particular zone is. This is useful because we can either create a white-list or black-list for specific types of land use.

I used a blacklist with ”grass”, ”parking”, ”farmland”, ”fountain”, ”fuel”, and ”lavoir”. ”Farmland” is not to be expected inside of a city, while the other types are physically too small to be of importance. For example, if someone visits a fountain, they are most likely visiting the park that the fountain is in and not the fountain itself. Similarly, while people definitely use parking lots, the parking lot is not their actual destination.

As the OSM map contains land use tags for each part of a city, we can simply assign destinations for the agents from the pool of locations. So we could take a residential area, a commercial area, and an industrial area and combine them to give the destinations for the day.

While the OSM map can easily be parsed for various land uses, since we are collecting all possible land use tags one issue that appears is that we also get lots of physically tiny tags. For example, fountains and vending machines are classified with the ”amenity” tag (simply dropping all ”amenity” tags is not possible as ”university” and ”school” are also classified as such). Typically, people do not travel to only a vending machine. Since we are collecting all of the tags we must define a white/black list or deal with them in some other way. However, as we are going to define specific location types for the agents (a whitelist), we do not need to worry about them in this particular case. Nonetheless, it is something to keep in mind if more randomly selected agents were to be used.

For generating generic agents, we can randomly assign them destinations from anywhere but as previously explained some destinations will be ”junk” (like vending machine) while others may not make contextual sense. While some people may move from a residential zone to a residential zone (house cleaners or delivery), most typically go to a commercial / retail / industrial / etc. zone to work. So we should expect that most of the agents generated move between a residential and commercial zone and not two residential ones. However, with a uniform random sampling, this will not happen as most locations are residential and thus most combinations will be residential to residential. Commercial to commercial also does not make sense (the person would be living in a commercial zone). Unfortunately, this can be complicated as lots of buildings in older parts of a city are mixed use, with the ground floor commonly retail and the upper floors residential. OpenStreetMap tags these buildings as residential meaning that it may be more acceptable to see residential to residential movements.

One way in which we can create agents is by defining groups of similar people that we would expect to be traveling around each day. A typical worker could be defined as moving from their residence to a commercial location and then back again at the end of the day.

A student would be similar to a worker in that they move between their residence and a ”work” location and back. In this case, the work location would be a university or college. While some students may also hold a job in addition to education, these cases do not need to be considered for a ”standard” student.

A parent could be defined as a worker who also has to drop off and then pick up a child before and after work. This means that instead of having a single destination (besides returning home at the end of the day), they also have two intermediate stops at a school. As most people fit roughly into these categories, we can assume that any variation beyond these is atypical and not needed for a representative group of agents.

Just as important as the locations themselves, the agents should be given appropriate times as to when they should go to each of these locations. Randomly selecting between a two hours should be good enough, but several questions arise. Should the sampling be uniform or from some distribution (such as normal) as well as what should the hours themselves be? Additionally, should we assume that all of the businesses in an entire city open at once or do they stagger themselves in some way? Presumably some businesses such as service based ones will be open later in the day (personal care, groceries, etc.) in order to serve other workers whose day begins earlier in the morning. Similarly to how the agent types are roughly defined, we can randomly and roughly assign times to agents.

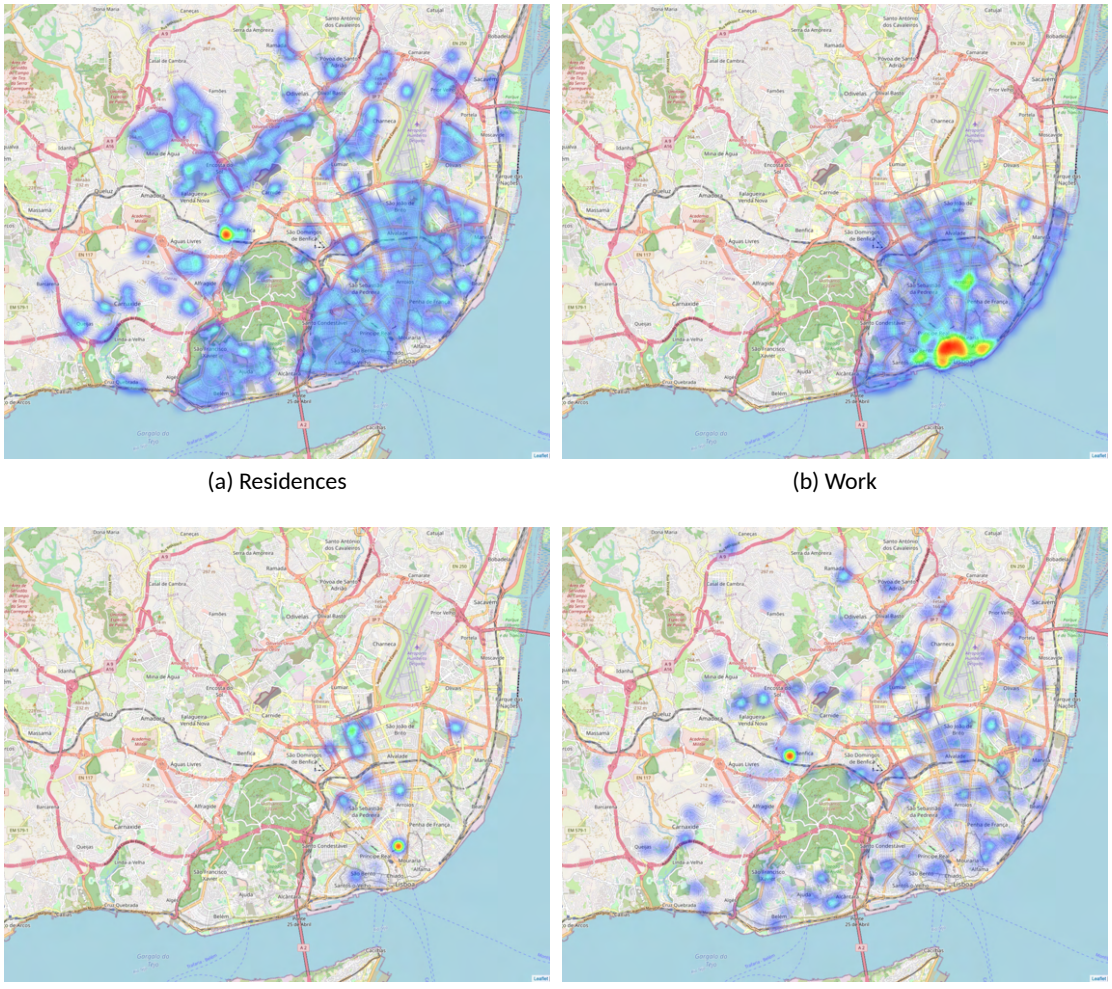
After we define a selection of agent types, what then is the best method to sample from them in order to get a representative population? How should the locations and time of day also be sampled? Since the goal was not to necessarily generate perfect agents, but rather to generate plausible enough ones, not too much time was spent perfecting the sampling. With a very large number of agents, getting each individual exactly right also becomes less important.

Finally we can consider the applicability of including additional daily activities such as shopping. Should these agents include a shop as another destination or should we make the assumption that they shop close enough to their homes such that that extra trip will not make any difference to a macro view of traffic?

As stated, about 590,000 cars move around in Lisbon each day with 220,000 originating in the city while the remaining 370,000 originate outside and move into the city. With these numbers, 590,000 agents were created with their main destination inside the city while their residence is either inside or outside (based on the 22 vs 37 ratio). The agents were then given a main type based on the following percentages: 0.845, worker; 0.15, parent; 0.005, student.

Workers were given a random work location from all of the locations with one of the tags: ”commercial”, ”industrial”, ”retail”, ”construction”, ”restaurant”, and ”cafe”. Parents were also given a job from the aforementioned list as well as a school location. The school given is the geographically closest to the agent’s residence. Students are given a university rather than a workplace.

The distribution of these can be seen in Figure 4.1. As the plot is a heatmap, a warmer color means that more people are in that location. One issue is that since the locations were uniformly distribution regardless of physical size. Smaller locations are equally likely to be chosen as larger ones and thus smaller ones will appear to be warmer (more popular) as agents are more ”physically” overlapping. This also means that businesses with fewer employees are equally as likely to be chosen as businesses with more.



(a) Residences

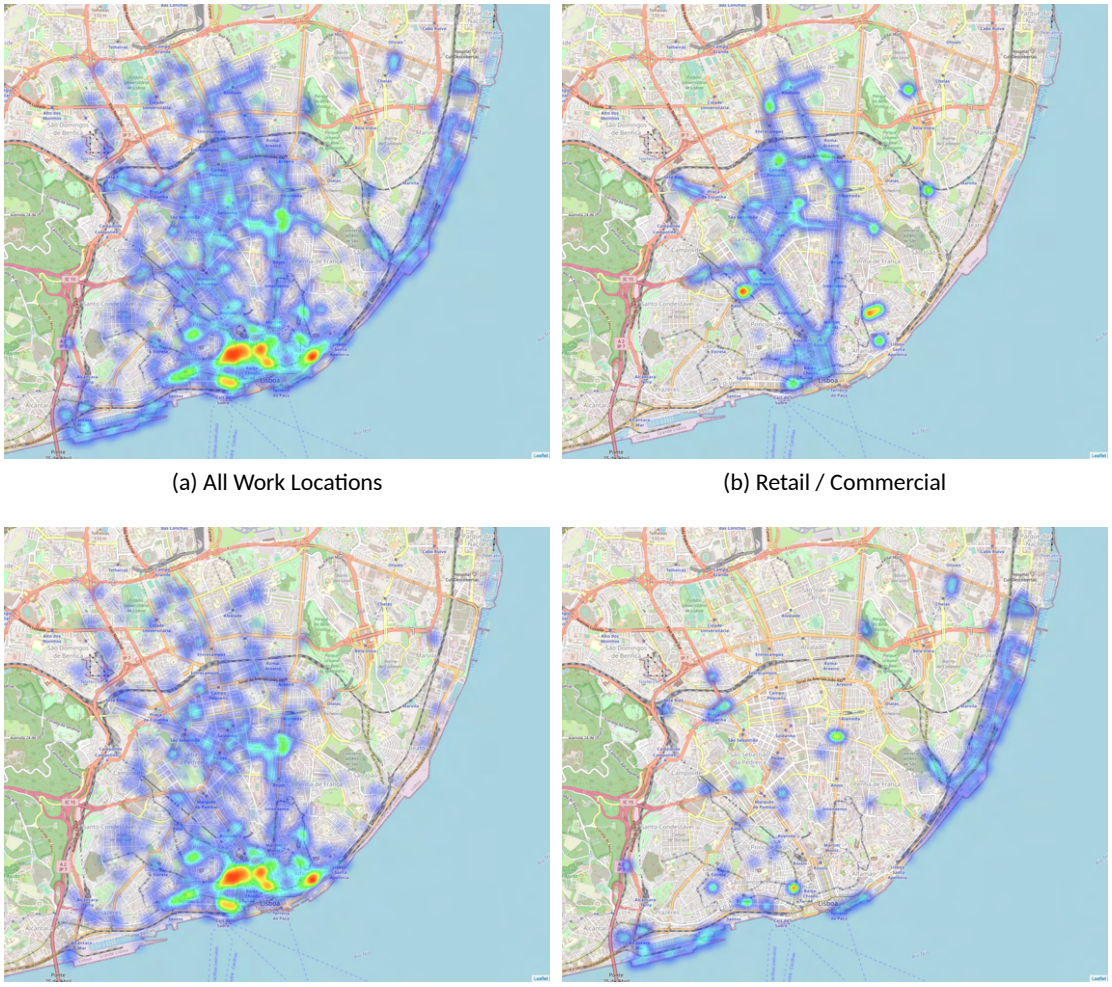
(b) Work

(c) Universities

(d) Schools

Figure 4.2: Generic Locations

In Figure 4.2 we can see the different distributions of home and work locations. For example, we can see that downtown Lisbon has more work locations (Figure 4.1b) than residential ones (Figure 4.1a) while in the Ajuda and Restelo neighborhoods there are many more residences than work locations.



(a) All Work Locations

(b) Retail / Commercial

(c) Cafes / Restaurants

(d) Industries / Construction

Figure 4.3: Different Work Locations

In Figure 4.3 we can see the different distributions of different work types. In 4.3a all of the work locations can be seen together, while in the other subfigures we can see the other types of work locations. 4.3b shows retail and commercial areas, which are mainly focused around the downtown area. 4.3c shows cafes and restaurants, which while they are somewhat evenly distributed throughout the city, most are centered around downtown. 4.3d shows industrial areas, which are mainly around the water and outside of downtown.

Chapter 5

Traffic Simulation Model

As we have already discussed the method in which the agents will be created, we can now focus on the simulation itself. As the simulation can work with any map, the agents need their routes to be produced with regard to the current map and not generically.. Therefore at the beginning of every simulation, the agents' routes must either be calculated or loaded from a previous simulation of the same map.

While it may be possible to let agents perform a random walk to calculate their routes, it would be computationally very difficult to find the destination. If we assume that at each road there are 3 possible roads to continue onto, we find that there are, as an upper limit, 3^N with N the number of roads between the origin and destination. So if we assume there are roughly 20 roads, between the start and finish, this means that there are over 3 billion possible routes. Even if we remove routes that visit the same roads more than once, it should still be a very large number. Considering that the roads (for the city of Lisbon) average around 100 meters long, this means that a 10 kilometer drive should pass through around 100 roads or $3^{100} \approx 10^{47}$.

In order to solve this problem, we can use the A^* algorithm with a graph of the road network and a specific weight matrix (this matrix will be explained in more detail later). As this weight matrix assumes empty roads with no delays (traffic, stop signs, or stop lights), the routes generated with it are, in theory, the fastest possible. Because the matrix does not consider how speeds will change in reality, we will use the simulation for this purpose.

Once the agents have been defined, the next step of initializing the simulation can begin. As we already have defined the area of interest when generating the agents, we can easily download a map from OpenStreetMap (OSM). Once the map file has been downloaded an external package OpenStreetMapX (OSMX) is used to parse the map data and extract the road data. The OSM data also includes lots of other information concerning buildings and points of interest. While like in the real world, this extra information may not directly be influencing traffic itself, but it may be used to influence the creation of the agents (like land use).

The output of the OSMX package is a graph of the road network and several matrices (most importantly a weight matrix that describes how long each road is in meters) that describe some information for the edges (roads) between the vertices of the graph (intersections). From this road data, we can derive a lot (but certainly not all) of the information of each road. The limitations of using OSM is explained in section 7.1. The information that we will use from OSM is the length of the road, its class, and whether it is a one-way street or not. The class is defined as the type of road a road is. As there are 8 classes, the class differentiates whether a road is a motorway versus a main road versus a residential road. We can then use the class to assign different values for the speed of the road, how many lanes it has, etc. This simplification to just a few classes is certainly guaranteed to have exceptions, which are explained in section 7.1.

In the simulation, each road is defined by an object that holds various attributes of that particular road. These include static things such as the speed, length, vertex ids, and vertex latitude/longitude coordinates. There are also a few attributes that are derived and/or possibly manually set by the user. For example, from the OSM data we will have defined the number of lanes and length for each road. Using this information we can derive the total number of cars that can fit on each road. OSMX creates ”roads” by breaking a road into road segments between each intersection. This means that even though a road in real life may pass through many intersections, here each segment will be considered a separate road. The maximum number of cars is simply the total number of cars that can fit on a road at one time.

The way in which this is achieved is by using a queue system for each road. Each road has a queue that is filled front to back as cars are added and is emptied front to back as cars leave that road and move to their next one. Deriving this maximum number correctly is quite important as having a number that is too large will result in very few, if any, traffic jams (not ideal if we are trying to see where these jams occur). A number that is too small will result in extreme traffic jams that may never resolve themselves. For example, as some of the roads can be shorter than a single meter, if we were to simply say that the max number is the length of the road divided by the average length of a car (3 or 4 meters), then we would (rounded to the nearest integer) get a maximum value of 0. This would mean that no cars could ever move onto this road, and therefore continue to the next roads. Even if we are to set it to 1, this can create problems as having a single car on a

road, will by definition be jammed, as it is completely full. This will cause traffic jams in previous roads as the tiny one must always be completely empty before any other car can move there. So while this is not a problem for large/long roads, we are tasked with finding a minimum value (that is greater than 1) for the tiny roads. After some testing, a minimum value of 4 seems to work well.

As this simulation does not implement traffic lights, each intersection is basically assumed to always have a stop sign. While this is not true for many major roads that would have stop lights, as the agents are only stopping for a brief moment, we can assume that the end result should be similar.

Finally, the most important task for finding the agents' daily routes is defining the weight matrix for the graph. This starts initially as a distance matrix (for each road segment between each intersection), and then is divided by the speed of that road in order to turn it into a time matrix (time to cross the road). Using the time is the best as it does not bias towards just the fastest (speed wise) or shortest (distance wise) roads, but rather the fastest (time wise).

The simulation itself is relatively quite straightforward once the road network and agents are fully set up. The simulation repeats for multiple days with each day being different, yet coupled, to the last. Each day is broken up into timesteps, with each day lasting for as long as needed. For these simulations, I let it run from 00:00 to 27:00, with hour 27 representing 3:00 of the following morning.

The timestep of the simulation may be as small as a single second. While a second is most accurate and still runs quite quickly, a larger timestep may also be used to speed up the simulation (I was averaging around 20-30 minutes per simulation day with 6 cpu cores). However, if this number is too large, it will introduce a major issue. For example, if a road is 11 seconds "long", then a timestep of 1 second will take 11 seconds to cross whereas a timestep of 10 will take 20. This is because at 10 seconds the agent has not yet crossed the road and must wait for the next iteration 10 seconds later. It effectively doubles the time spent on that the road even though, which means traffic must wait more time before moving, causing delays which may lead to unrealistic jams.

After this parameter is set, the basic simulation for each timestep is as follows, with a longer description following the list:

1. Iterate through all agents to see if any have their departure time equal to the current time. This is achieved by simply checking all of the agents' departure times with the current time and then for each agent where it matches, placing that agent on its starting road.
2. Iterate through all roads:
   (a) For each road, the simulation will check if that road has agents on it. If so, it will run through the queue, starting at the beginning of the queue.
   (b) For each agent in the queue, 1 timestep should be added to their personal counter.
   (c) Each road has a maximum (the maximum number of agents per timestep is set by the class of the road). If an agent fails to move forward then the maximum number becomes that agent's number. If an agent is unable to move forward, any agents behind them will also not move forward. For example, if the maximum number is 4 and it is currently the first agent in the queue, then the following steps should be checked. If it is the 5th agent ($5 \not< 4$), then the following steps should be skipped for this agent (and by extension all following agents in the current road).
      i. If the agent's time counter is higher than the road's length, in time (time = distance / speed), then the agent should check the next item. If not, then no more steps should be checked for this agent, and the next one in the queue should be checked.
      ii. If the agent has moved to its next road then it will record the total time it took to cross that road. It will also be put at the end of the queue of that next road.
      iii. If an agent has waited on a road for a set number of seconds (Irregardless of any delays or any other reason to keep the agent from progressing), the agent will "teleport" to its next road.
      iv. If the agent has gotten reached its second to last road on its route, then the agent has finished its route and is taken out of the simulation. The agent may re-enter, as described by step 1, if it has more routes to complete on the same day.

The movement between roads is naturally complicated, as the road ahead must be considered as well as the order of the current queue. The purpose of the maximum value is to keep agents from passing each other. While this is a reasonable requirement for small one-lane roads, it is not for larger multi-lane ones. Nonetheless as the queue is ordered, only the first agent in line is allowed to move to the next road, as long as the next road's queue is not full. Each road has a maximum size of its queue and so as long as the current size is less than the maximum size, it is considered not full.

If an agent successfully moves, the second in line agent is allowed to move (as long as they also have their next road open and they have been on the current road for long enough). This repeats until either an agent fails to move (so every agent behind them would also be stuck) or the maximum number of agents has been reached.

The maximum number of agents that can move for each timestep is dependent on the size of the road with larger roads (larger in width and speed, not necessarily length) naturally allowing for more agents to move at once, such as a multi-lane boulevard compared to a single-lane alleyway.

The rule of stopping the remaining agents when an agent fails to move is only realistic for a single-lane road as multi-lane roads may have lanes that can (or only) turn or go straight as well as the ability for cars to temporarily go into another lane to pass a stopped car.

After an agent crosses a road, it will record the amount of time it took to cross that road in a counter. This global counter adds the time it takes for all agents that cross that particular road, irrespective of where any of those agents are going. Dividing this counter by the total number of agents that have passed will give us the average amount of time it took to cross that road. As this average time will change at different points in the day, we will have a different counter for each 15 minute period of the day. So for a 27 hour simulation, there are 108 values (of the time it takes to cross) for each road. These values will be very similar for certain times of day while also being quite different at different times. The choice of using a time period of 15 minutes for different counters is flexible as different times are required for differently sized models and different modes of transportation.

If an agent has been in the same location for more than a set amount of time, it will be forced to the next road. This ”teleportation” parameter can be helpful as there may be some jams that will not be able to fix themselves otherwise. While this is not perfectly realistic, it can help fix some of the issues of using OSM maps (section 7.1).

After the agent reaches its destination, it will ”disappear” from the roads. If the agent has multiple routes in a day (multiple activities that have a break in between them), the agent will have its current route set to that next one and will begin that subsequent route in the same manner as its first (see Step 1).

Algorithm 2: Timestep: Runs the simulation for 1 timestep

Input: Current simulation time; A list of agents; A list of roads; Various weight matrices
Output: A list of agents; A list of roads; Various weight matrices

```
for i ← 1 to number_of_agents do
    if agent_i.departure_time equals current_sim_time then
        Place the agent on its first road
for i ← 1 to number_of_roads do
    if road_i.queue is not empty then
        for agent in road_i.queue do
            if agent is allowed to move then
                Move agent to its next road
return nothing
```

Figure 5.1: Pseudocode of a Timestep of Traffic Simulation

At the end of each day, all of the agents will have finished their routes and some of the agents will have their routes recalculated. This is achieved by using the A^* algorithm to calculate the newest shortest route between each of the selected agent's destinations. The weight matrix used for A^* is no longer the theoretical one used in the beginning, but instead one of the 108 time counter matrices created during the simulation. Selecting the one that matches the agent's departure time will give a much better approximation of the road conditions at that time of day than just using the theoretical times. This will also allow agents to select routes that may be slower in theory, but can be faster at certain times of day as they are less crowded.

10% of the agents that took the longest to complete their routes have it recalculated every day. Ideally, after so many days, the weight matrices would find a global minimum and each agent would be taking the best global route.

As any simulation can always be improved to make it more and more realistic, here are some of the possible improvements to this simulation.

In this simulation, each road is defined by its class. This means we can understand whether it is a motorway or a tiny farm road. However, we could also add more information to the road to change how the agents perceive that road. For example, the slope of a road is perhaps not that important for the average driver but very much so for a bicyclist. Another example would be if the road has street parking. This means that drivers on this road may stop much more frequently as they look for and find parking spaces. More qualitative metrics could also be used, such as the beauty or perceived quality of a certain road over another.

Currently, the cost function used to recalculate an agent's route is just the total time taken for each route. If more road attributes are added, then the cost function should change to take these into account. For example, take two different routes that take the same amount of time, one of which is physically longer while the other has lots of traffic jams. Even though they are identical time-wise, the cost function could potentially be written such that it prefers more comfortable journeys (a longer no traffic route over a short jammed one). This would be especially important for other modes of transportation. Bicyclists (and pedestrians to some extent) have to be concerned with the slope and quality of the road, while people on public transportation could be concerned with how full a particular bus is.

At each intersection between roads there must be some way of controlling which cars have the right of way. For smaller intersections, a simple stop sign (or assumed one) is sufficient. But stopping at every intersection is slow, especially if lots of cars need to take turns moving through it. For the simulation, adding stop lights would help traffic flow for the major streets in a city as agents could move quickly between several intersections if they are ”lucky”. If the agents are not lucky, they may also have to wait a significant amount of time between each intersection (such as at a red light) which is also realistic. Unfortunately, as the OSM map is not perfectly consistent with this information, we cannot assume that it is perfect or complete enough to be usable in all cases.

For most small side streets, using a single lane is realistic and sufficient. For larger streets, this may not be the case. Depending on whether agents are allowed to stop for other reasons than for traffic jams, a multi-lane road could add much more realism to how cars can flow. For example, if a car is parking or a delivery truck / bus stops, then cars behind it can go around by changing lanes. However, if those features are not looked at, then a multi-lane road may not change the final results compared to using a single lane road. In the case of this simulation, the single lane roads have their capacity increased to account for the extra lanes, even if the lanes do not really ”exist”. The other use for multi-lane roads is to have specialty lanes. So some lanes may be only for turning. For example, if there is a road that has a traffic jam on the road straight ahead, but not the side street intersecting it, the turning lanes could still allow for traffic flow despite the straight lanes not allowing it.

The output of the simulation depends on how the agents are defined. The agents can either be set up with a single route/destination for each day (trip type) or with multiple routes/destinations (activity type). If it is the former, then less specific information from each agent can be extracted. This is because when the trips are not connected, information such as total distance/time traveled for each agent in a single day is not possible to collect.

Information that is possible to collect from the agents is the distance and time traveled for each single route, and the amount of time or times stuck in a traffic jam. Information from each road that can be collected is the number of jams (and when those jams occurred), and the average amount of time taken to cross that road (for different times of day). The amount of time taken (stored as a list of matrices) can even be used as the input for another simulation (which we do later).

Global information can also be extracted such as the average amount of time spent on each road as well as the number of cars that have crossed that road. Since this information can change throughout the day, I have decided to record this information at 15 minute intervals. This is a good level of enough time such that the values will be a good average while also still being quite frequent. Since the following chapter's results focuses on global values of the whole traffic network, only the time spent and number of cars passing on each road is used. These values are stored as an array of matrices with a matrix accessed by the starting and ending vertex of the desired road.

Chapter 6

Results & Discussion

For these results, the 590,000 generically generated agents from chapter 4 were used.

Once the road weight matrices have been extracted, they can be plotted on a map in order to see how they change over time. One way of plotting this information is to use a heatmap, with roads with more traffic / slower times having a redder or darker color. ”Faster” roads can be greener or a lighter color. One of the issues of this, however, is how to normalize the time between the different roads. If the raw time is used, then larger roads will naturally have a larger value and therefore they will be represented by a darker color and the reader will assume that means that those roads have traffic jams.

One way of normalizing the data is to subtract the times by their theoretical minimum times. This would result in an absolute difference between the real and theoretical times. In testing, this also seems to affect longer roads as these will naturally have larger absolute differences in between the real and theoretical times. Another way to normalize the data is to then take the difference between the real and theoretical and divide that difference by the theoretical. This gives us the relative difference in time. This seems to work much better as it gives a percent difference in time rather than actual difference in time.

Unfortunately, due to the issue with very short roads, using the theoretical value does not work very well on its own. As the theoretical value assumes that the timestep is infinitely small, the theoretical value does not take into account rounding by the use of any sized timestep. For example, a theoretical value of 0.5 seconds will take at least 1 timestep (plus any delay between roads that may be added). If the timestep is set to 1 second, then the relative value is 0.5 divided by 0.5 or 1 second and that means that relative difference is already 100%. While this may be

fixed by using extremely short timesteps, it is computationally very expensive as it is not a very widespread issue (although it still is quite major). So in order to get around this issue, we can force the minimum road length to be equal to the size of the timestep. So with a timestep of 1 second, the minimum ”length” of any road is 1 second. Distance (meter) length is not the same as the time (second) length. As this is just manipulating the time taken, this step can be performed before or after the simulation as needed.

Plotting the road times in an efficient manner is fundamental. After the simulation runs, the average time for each road for each 15 minutes of the day is returned as an list of matrices. With the matrices a heatmap (made with the Folium package) can be created. As the heatmap only creates circular points on the map, and not straight lines, I placed a point every 20 meters along a road to simulate a line. This seems to be an appropriate value as only placing a single point, for every road, is confusing and hard to visualize while placing many points for a road (to make it seem continuous) is very strenuous on the computer (doubling the number of points means double the computation time for each heatmap and double the final file size).

The heatmap also requires values to be between 0 and 1. If values are outside of this range then they may default to a value of 1 (which can cause problems as it appears values are the same even if they are not). Because of this, I had to scale the results to fit this range. For the delay, a relative value of 0 means that the actual time is the same as the expected time. This means that the best possible scenario is 0 while a value of 1 represents a delay of 100% or double the expected time. Any delay longer than this is clipped to 1. Scaling more to allow for longer delays to be displayed would also have the negative effect of making shorter delays harder to visualize. For the figures that show the total number of cars, every value is divided by the maximal value recorded on that day. This means that every value displayed falls between 0 and 1.

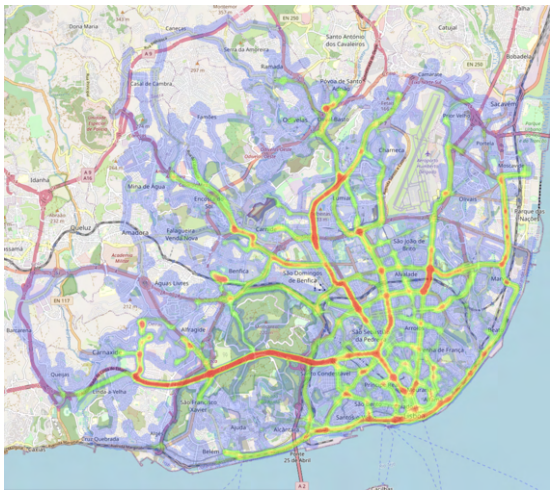
6.1 The Simulation Results

The first simulation that was run was for 20 days to see which routes the agents would learn to be the fastest. The agents had no prior knowledge of which route is best or how much traffic there would be on each road at certain times of day. They only knew the theoretically fastest route and used that route for the first day. On subsequent days, the 10% worst performing agents would reroute their route to see if a better one was available. The graphs on the left side show the values for day 1 of the simulation (with routes generated from the theoretical times), while the graphs on the right side show the values for day 20 (with routes generated from the results of day 19, which came from day 18, ...).

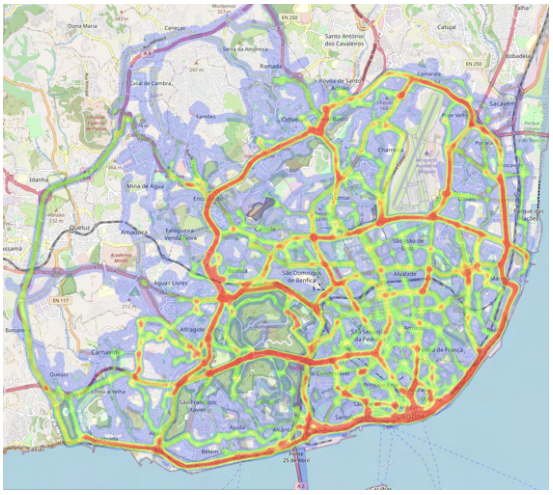
Number of Cars

The first way that we can analyze the results of the simulation is to simply look at the number of cars on each road at each time. This will give us an idea of where cars are during different parts of the day. The values were normalized by dividing by the largest value found over both the first and last days. This value, 5441, represents the maximum number of cars that were on any road during a 15 minute period. This value occurred during the first day at 19:30. Any road with a valueless than 0.1% of the maximum (a value of 5 in this case) were not included to save processing time.

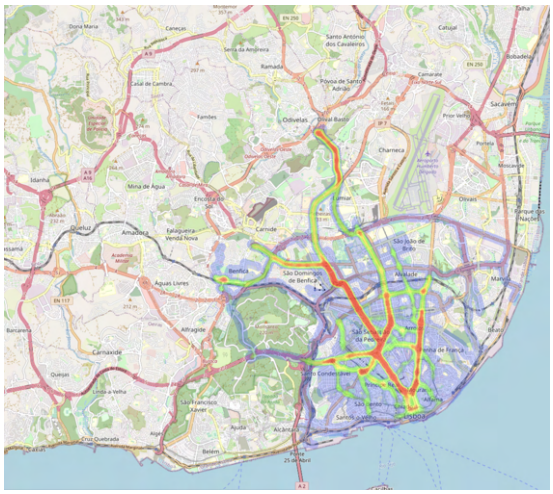
In Figure 6.1, we can see the progression of traffic in 4 hour increments. We can see the morning rush hour before a quiet period during the middle of the day before a return to traffic during the evening rush hour. We can also compare days 1 and 20 for each hour as the two days are side by side. While day 20 has more red roads (more traffic), this is due to the agents seeking new routes. So while more roads are crowded, the total number of agents has remained the same and thus the busiest roads should have fewer agents. For instance on day 1 most of the agents seem to favor traveling on ”Avenida da República”, while on day 20 the agents have moved to outer roads. Another indicator that the simulation is working correctly is that all of the agents have completed their routes before the end of the day on day 20. This means that while the traffic during rush hour is seemingly ”more intense” on the whole, spreading it out over more roads means that the agents can complete their individual routes more quickly.



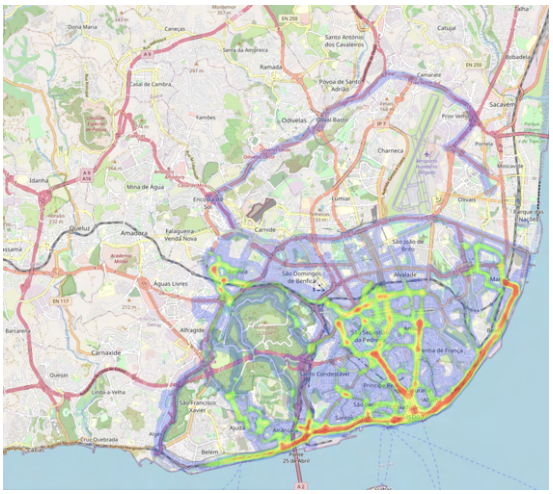
(a) Day 1; hour 8



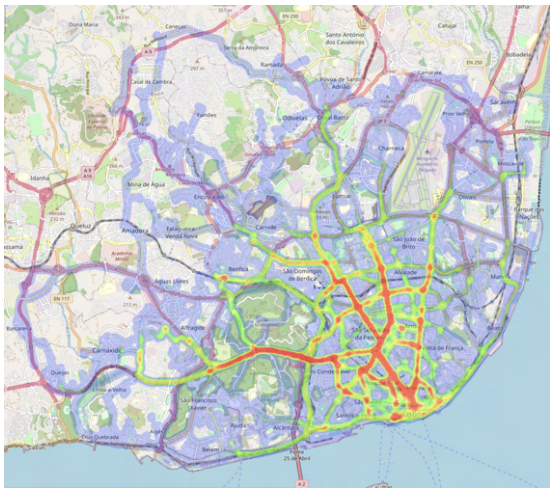
(b) Day 20; hour 8



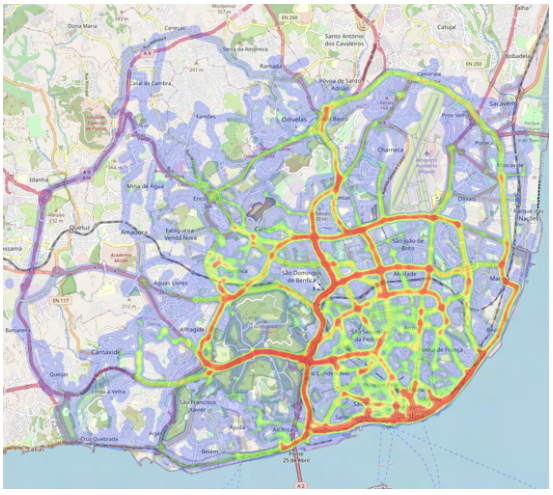
(c) Day 1; hour 12



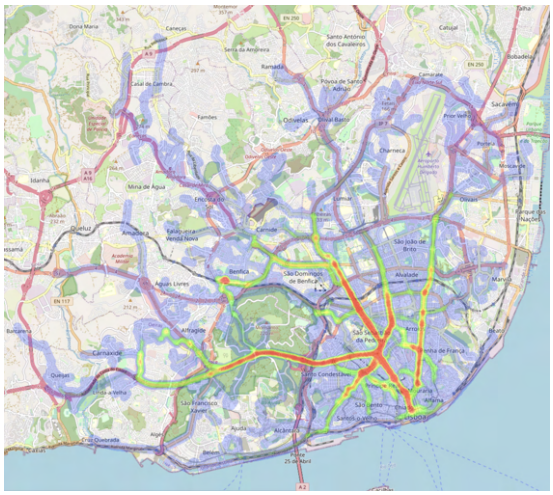
(d) Day 20; hour 12



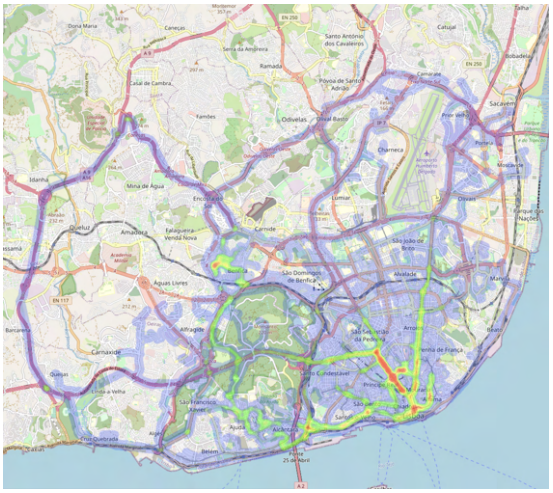
(e) Day 1; hour 16



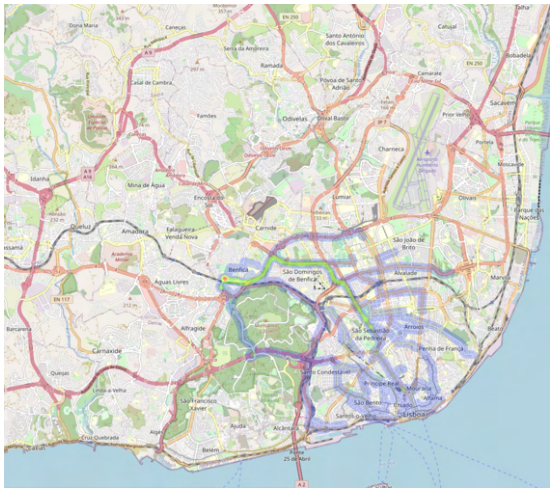
(f) Day 20; hour 16



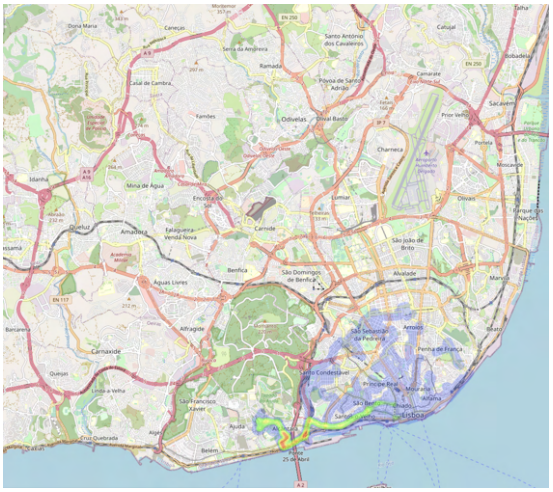
(g) Day 1; hour 20



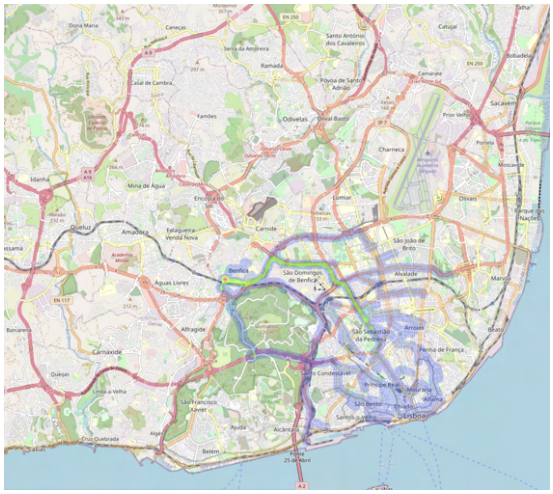
(h) Day 20; hour 20



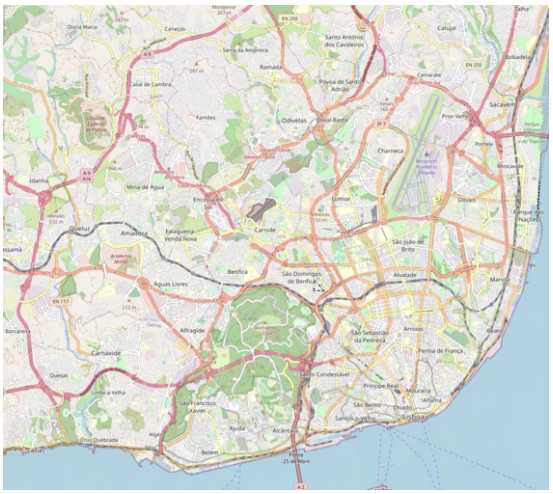
(i) Day 1; hour 24



(j) Day 20; hour 24



(k) Day 1; hour 26



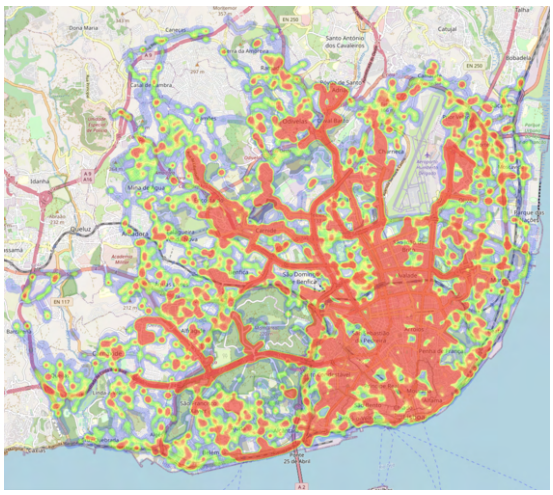
(l) Day 20; hour 26

Figure 6.1: Total Amount of Cars

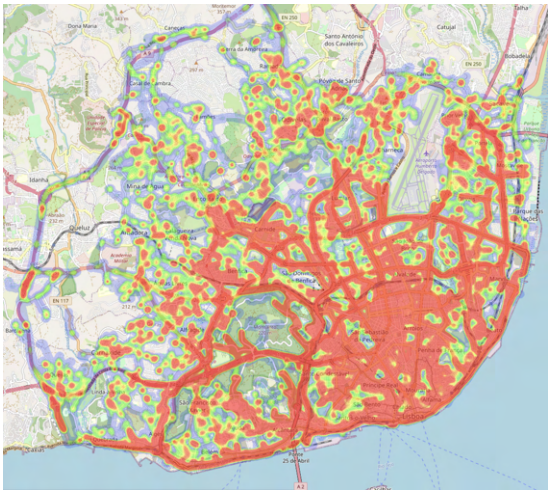
Length of Delay

Besides looking at just the number of cars, we can take a look at the average relative delay on each road. This value is then averaged for all the cars on a road over each 15 minute period.

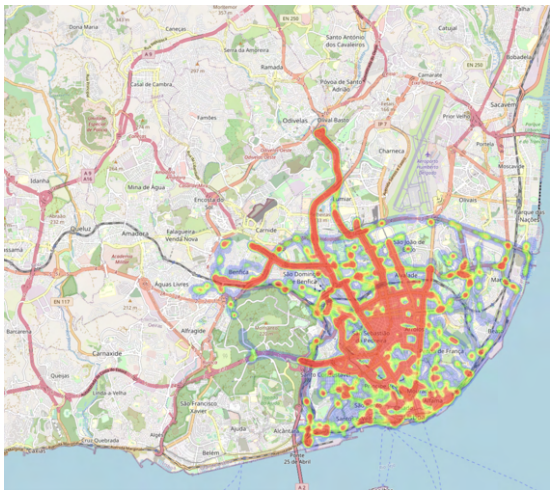
Looking at Figure 6.2 we can see that there is a lot more red on the heatmap compared to Figure 6.1. This is because delays can occur on smaller roads whereas smaller roads will simply not have the large numbers of cars seen on larger roads. Day 20 also appears to have more red points than day 1. This means that more traffic delays are occurring. Most of these points are on very short roads and thus suffer from the short road problem. More frequent, but shorter, delays should also be more favorable than fewer but longer delays.



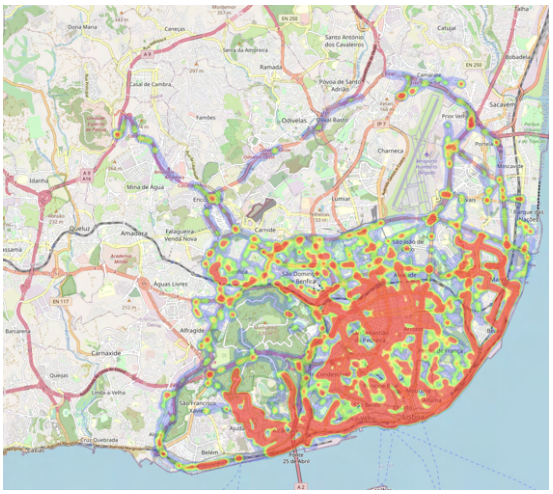
(a) Day 1; hour 8



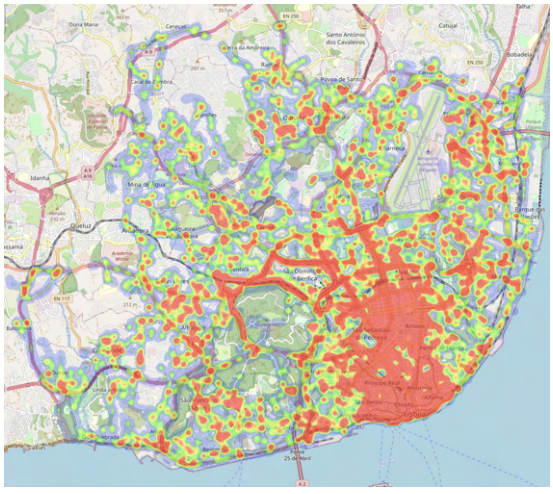
(b) Day 20; hour 8



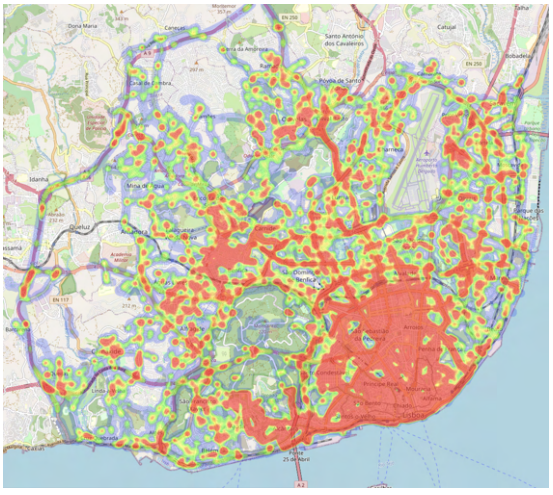
(c) Day 1; hour 12



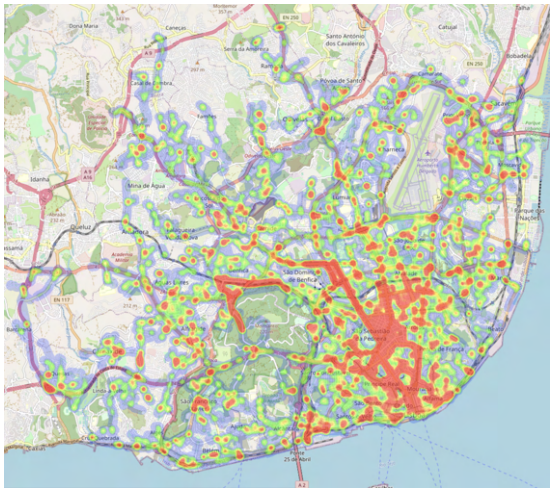
(d) Day 20; hour 12



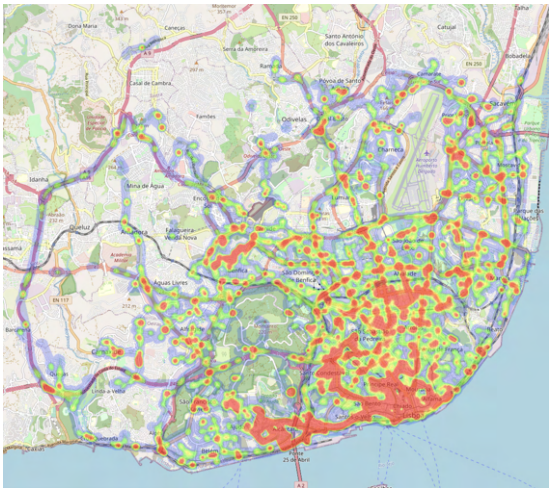
(e) Day 1; hour 16



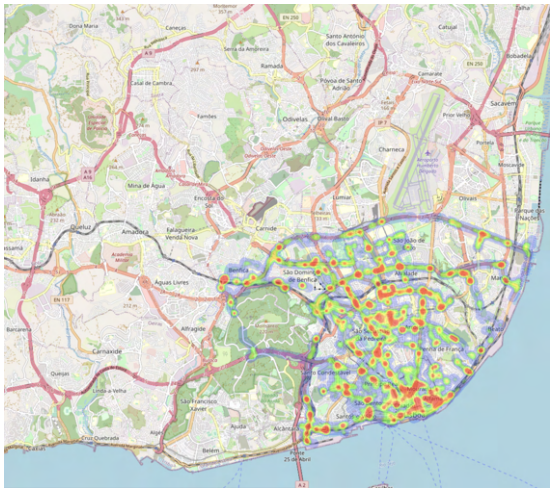
(f) Day 20; hour 16



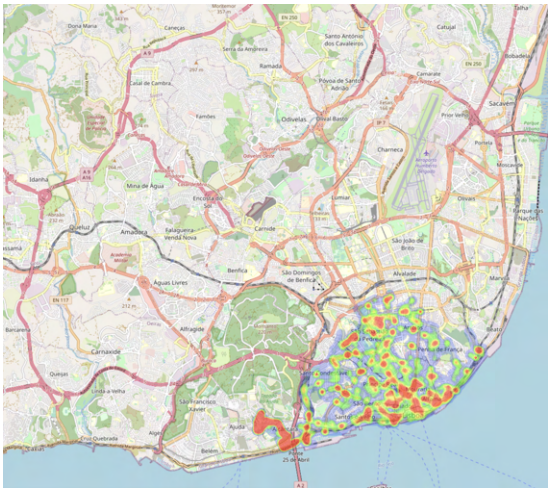
(g) Day 1; hour 20



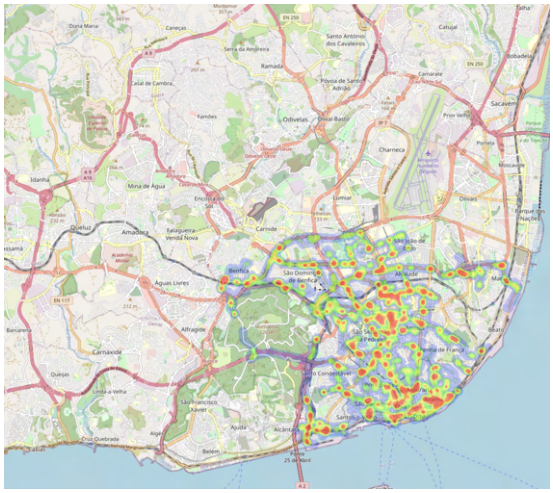
(h) Day 20; hour 20



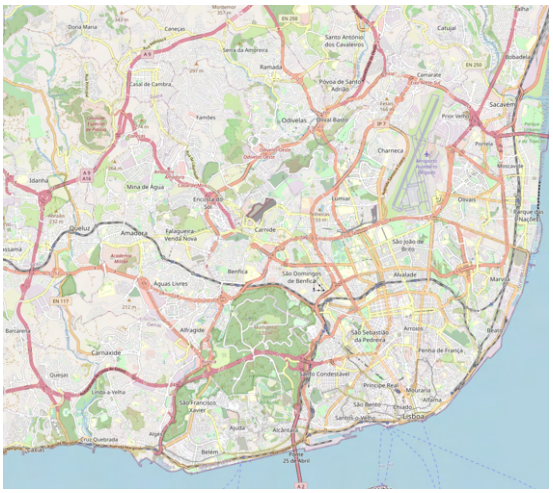
(i) Day 1; hour 24



(j) Day 20; hour 24



(k) Day 1; hour 26



(l) Day 20; hour 26

Figure 6.2: Delay Ratio

6.2 200 Days

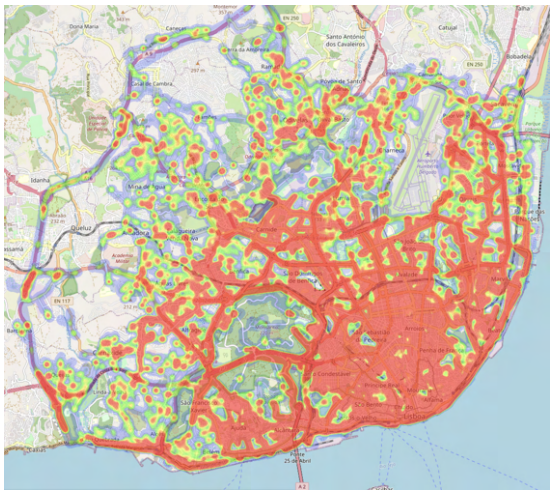
After analyzing a short run of 20 days, we will let the simulation run for 200 days. Letting it run for a longer time will allow the agents to settle into (hopefully) an equilibrium. This equilibrium should mean that the current routes for all of the agents will be the best when considering the other agents as well.

With the system in equilibrium, we will be able to perform some experiments, such as closing a specific road and seeing how the traffic flows around this closed road. By starting an experiment with a system in equilibrium, we can see its true neutral state. If we do not do this then we are limiting the usefulness of our results at the system is both adapting to our experimental change as well as tending towards the neutral equilibrium. Starting in a non-equilibrium also means that the results will be biased (such as how we initialized the system) towards the starting state.

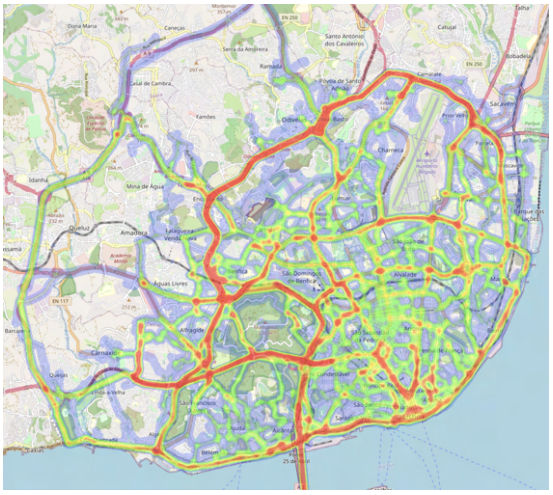
As with the previous 20 day simulation, the 200 day simulation will leave us with the average relative time delay on each road as well as the number of cars that passed. With the average time delay we can generate ”new” agents and start a new simulation. In theory this should mean that these new agents are identical to the agents on day 200 and that it is like the new simulation is running immediately after running it for 200 days. We may want to do this if we have to stop a simulation and resume it at a later time or if we would like to make changes to the simulation.

However, in practice, this did not appear to work. Looking at the heatmap generated from day 200 and the heatmap generated from day ”201” (creating all new agents from day 200 data and letting it run for 1 day more), it can be seen that the day 201 heatmap is much worse than day 200. For example, day 200 has no more traffic after 22:30, whereas the day 201 has traffic until the end of the simulation at 26:45 (2:45 of the following day). I suspect the reason for this not working correctly is that generating all 100% of the agents again causes the system to no longer be in equilibrium.

As the agents themselves were not saved after the 200 days, I made the decision to let the simulation run again such that the exact same agents (from day 200) could be used for further simulations. In this case the exact agents were saved and not just generated with data from day 200.

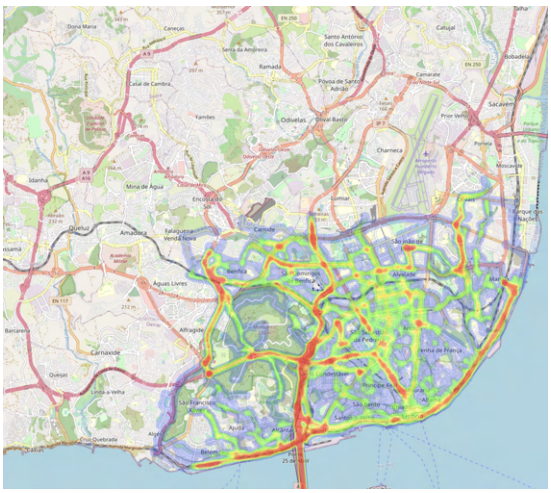


(a) Delay; hour 8

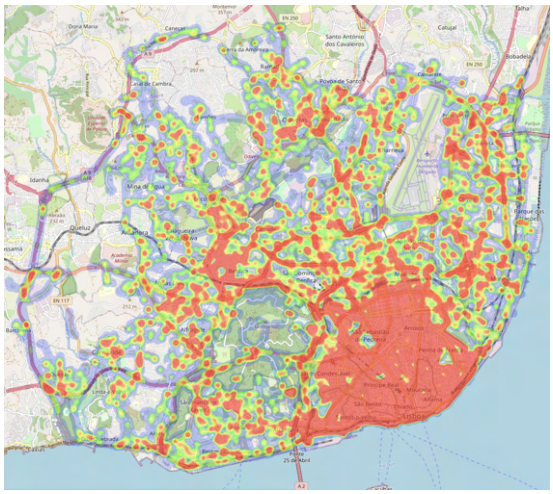


(b) Number of Cars; hour 8

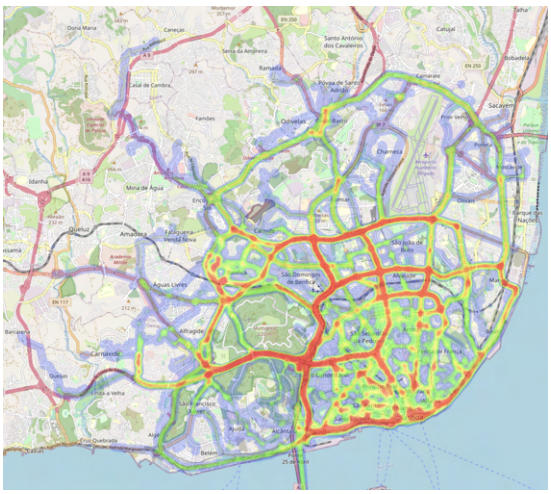
(c) Delay; hour 10



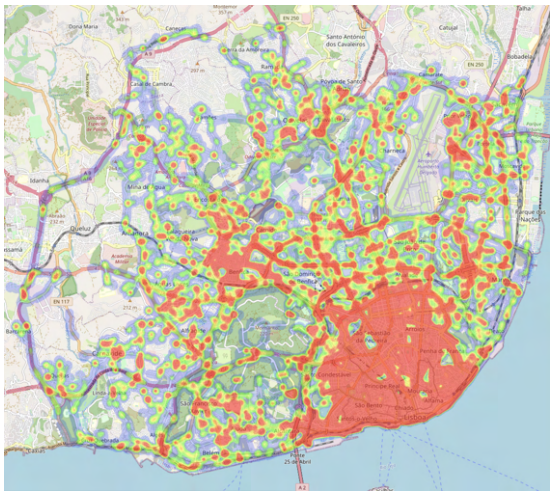
(d) Number of Cars; hour 10



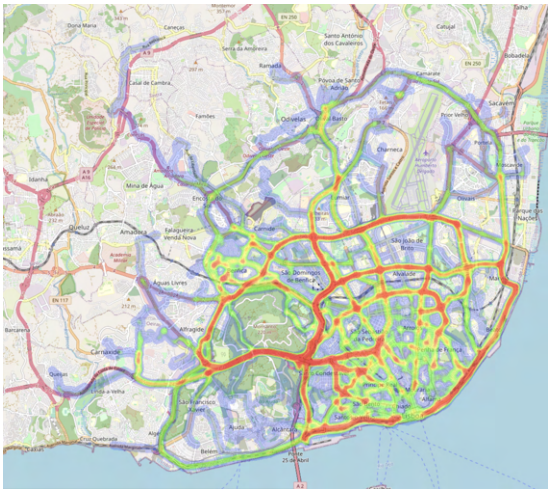
(e) Delay; hour 16



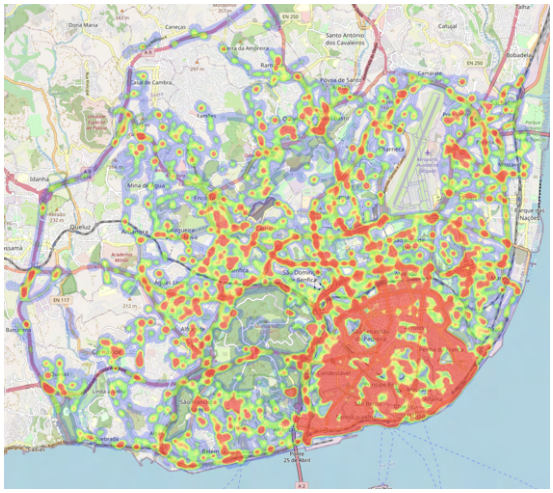
(f) Number of Cars; hour 16



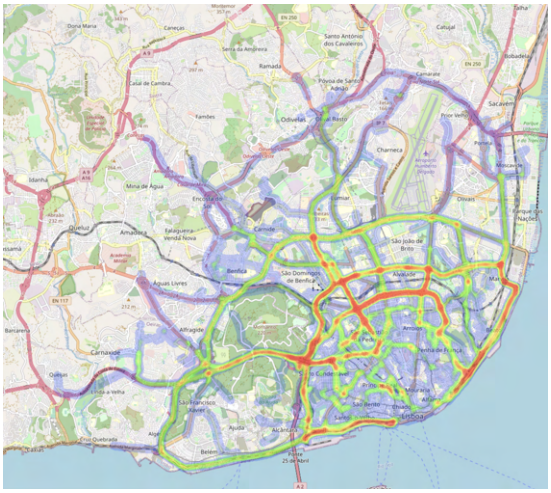
(g) Delay; hour 18



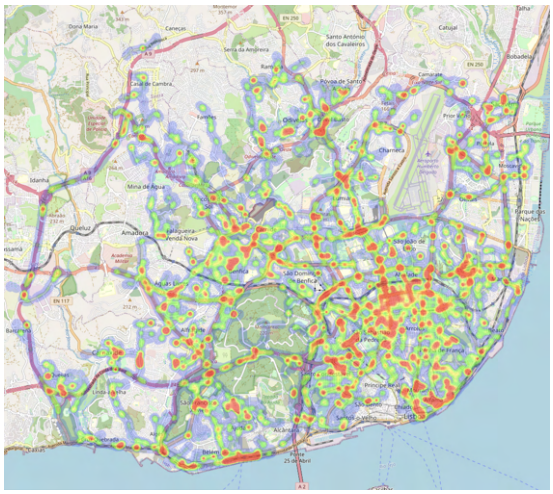
(h) Number of Cars; hour 18



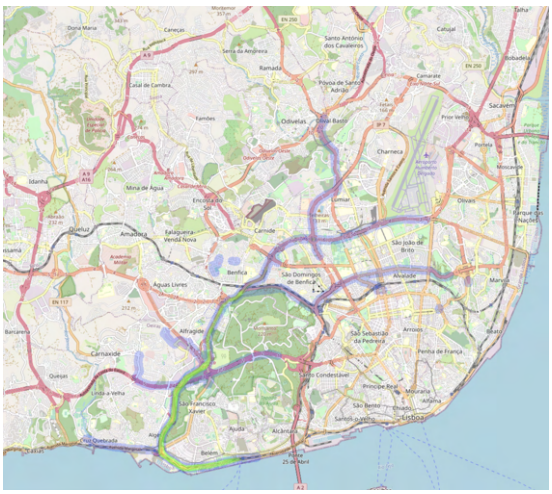
(i) Delay; hour 20



(j) Number of Cars; hour 20



(k) Delay; hour 22



(l) Number of Cars; hour 22

Figure 6.3: 200 Days

6.3 Removing a Road

One interesting use case for this road network simulation is the ability to remove roads and to see how the agents react. While the ability to do this for real-time events, such as a car accident, is preferable, it would require the simulation to become more complicated. For example, while cars immediately after the accident would have to reroute, so would cars that are several streets away. Should the cars farther away know about the accident, and if they do, when should they reroute? As this adds more complications, we can take a look at the simpler case of when a road is not accessible but it is known by drivers beforehand, such as construction. In this case, a road will not be accessible, but since the agents know this before they begin their routes, they will all plan accordingly at the beginning of the day.

One simple way of removing the road is to simply change its ”weight” in the average time matrix. If the weight is very high, the A^* will avoid that road and pick a new route. This does have the potential issue of still allowing cars to pass on the closed road if no other routes are found, but if this happens other problems would also occur.

As picking which exact road to remove is somewhat arbitrary here, we will see the effects of removing two distinctly different segments of roadway. The first is the ”Viaduto Duarte Pacheco” bridge (38.7223921, -9.1707868 to 38.7238127, -9.1784264). As it is the only (major) way to cross through the Monsanto park, it is a good choice to see how the agents will react to a major closure. The second choice is the second section of ”Avenida da Liberdade” between ”Rua Barata Salgueiro” and ”Rua Alexandre Herculano” (38.7232308, -9.1482020 to 38.7220723, -9.1472012). This is a major thoroughfare through the city center so closing it will certainly have an impact on traffic. However, since the road segment is in the city, there are lots of other options for agents to choose, unlike the bridge.

After the two roads were chosen, three simulations were run for 20 days each. Two of the simulations had one of the two roads ”closed”, while the third, the control, had no road closed. Even though the last simulation should not be any different from the 200 day one, it was run such that a comparison of closing a road would be the most ”apples to apples”.

6.3.1 Viaduto Duarte Pacheco

(a) Road Not Removed; hour 6

(b) Road Removed; hour 6

(c) Road Not Removed; hour 8

(d) Road Removed; hour 8

(e) Road Not Removed; hour 10

(f) Road Removed; hour 10

(g) Road Not Removed; hour 14

(h) Road Removed; hour 14

(i) Road Not Removed; hour 18

(j) Road Removed; hour 18

(k) Road Not Removed; hour 20

(l) Road Removed; hour 20

Figure 6.4: Viaducto; Delay Time

(a) Road Not Removed; hour 6

(b) Road Removed; hour 6

(c) Road Not Removed; hour 8

(d) Road Removed; hour 8

(e) Road Not Removed; hour 10

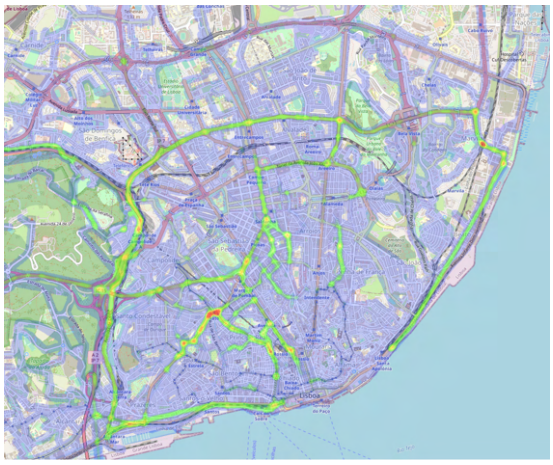
(f) Road Removed; hour 10

(g) Road Not Removed; hour 14

(h) Road Removed; hour 14

(i) Road Not Removed; hour 18

(j) Road Removed; hour 18



(k) Road Not Removed; hour 20

(l) Road Removed; hour 20

Figure 6.5: Viaducto; Total Amount of Cars

From the above Figure 6.4, we can see that closing the Viaducto may cause more traffic jams on the roads connecting the city center with the western suburbs. In the early morning, the roads north of Monsanto Park seem to be much more jammed although by mid-morning the control is also just as jammed. Interestingly during mid-day, the control seems to be more crowded south of Monsanto than the road removed one. In the evening, the IP7 road and Avenida das Forças Armadas become much more crowded for the road removed simulation than the control. This last point is especially strange since the closed section of road does not prohibit cars from exiting the IP7 and driving west along the Viaducto.

At first glance when we compare these to Figure 6.5, the figures do not seem to match very well. In all but the first hour shown, the IP7 road for the control has a lot more traffic than the corresponding one with the road removed. However, I suspect that some of this is a result of how the total number of cars is plotted. Since dark red represents the maximum number of cars that have passed on any road for the whole day, the exact number of cars for a certain color is different between days. Additionally, if a jam is very extreme and thus wait times are very long, the amount of cars that pass through that road should be lower and then the amount of cars is less. So in some cases, the delay time and amount of cars is inversely related. Too long of delays lead to less cars per hour than expected.

6.3.2 Avenida da Liberdade

(a) Road Not Removed; hour 6

(b) Road Removed; hour 6

(c) Road Not Removed; hour 8

(d) Road Removed; hour 8

(e) Road Not Removed; hour 10

(f) Road Removed; hour 10

(g) Road Not Removed; hour 14

(h) Road Removed; hour 14

(i) Road Not Removed; hour 18

(j) Road Removed; hour 18

(k) Road Not Removed; hour 20

(l) Road Removed; hour 20

Figure 6.6: Liberdade; Delay Time

(a) Road Not Removed; hour 6

(b) Road Removed; hour 6

(c) Road Not Removed; hour 8

(d) Road Removed; hour 8

(e) Road Not Removed; hour 10

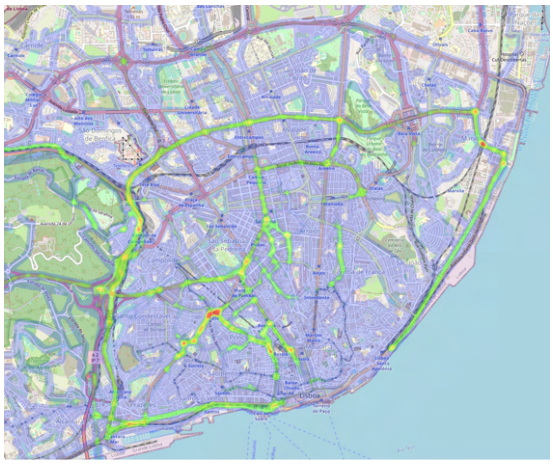
(f) Road Removed; hour 10

(g) Road Not Removed; hour 14

(h) Road Removed; hour 14

(i) Road Not Removed; hour 18

(j) Road Removed; hour 18



(k) Road Not Removed; hour 20

(l) Road Removed; hour 20

Figure 6.7: Liberdade; Total Amount of Cars

Compared to the Viaducto, the effect of closing Avenida da Liberdade is more subtle. While the delays (Figure 6.6) seem more or less the same between the control and the simulation, one big exception is the Viaducto! While that road is fully open for this simulation, the agents seem to favor it less as it requires going through the closed road. The lower levels of traffic can also be seen for the other roads that connect to Marquês de Pombal. While Avenida da Liberdade makes sense as it is the one with the closure, the other roads are used less as agents avoid the area altogether if their final destination is not there and they are simply passing through. For example, with Avenida da Liberdade closed it could make more sense for a driver to drive along the river if their destination is Praça do Comércio than for them to take the Viaducto and get stuck around Marquês de Pombal with no major road access to downtown.

In Figure 6.7, this conclusion is supported. The amount of traffic that passes through Marquês de Pombal is less in each of the hours displayed.

Chapter 7

Conclusion

The process of generating realistic human agents is naturally very complex as humans themselves are extremely complex. While many people have similar day to day activities, the randomness and variety of them makes it very hard to define a perfectly standard human agent that can then be copied many times for a simulation.

The method I used for generating generic agents is perhaps neither novel nor perfectly validated, but it did allow for quick generation of agents using no external data besides the map. Since one can easily download an OSM map for any part of the planet, this method allows for easy generation of agents that are ”good” enough for most simulations. Variations can be created by changing the white/black list of locations and definition of the types of agents.

The process for building a simulation from scratch was very long and complex as I had to continually fix and evolve the code. Despite these obvious drawbacks, by building it from scratch in a coding language that I understood, I was able to customize it to suit my needs exactly. I also understood exactly how the agents made each of their decisions throughout the simulation.

Besides being able to completely understand the code, doing it myself allowed me to design it as generically as possible. This means that the code is agnostic to the location and can instantly be run on a different location, anywhere from a different part of the same city to a different country.

The results of the simulation were interesting but not too surprising. The roads with the highest amount of cars are typically the larger circular roads around the city. Additionally, the traffic jams generated generally follow the expected peaks of the morning and evening traffic with a midday drop.

One major source of confusion in the results of the simulation comes from how the results were graphically displayed . The plots are unable to represent all the values at the same time without some sort of trade-off. Additionally the comparison between the delay ratio and the number of cars is tricky as they can both be correlated and inversely correlated at the same time. When a very small number of cars passes through a street during a given time, the delay may also be very small. If the number of cars rises then the delay may also rise as one would normally expect. However if the road is very small then the delay may be very high even if the number of cars is very small. This is also true for larger roads if there is a significant traffic jam. When there is a large jam, the cars will move slower and thus fewer of them pass through that road in a given amount of time. This means that while more cars can cause a higher delay, a higher delay can sometimes cause fewer cars.

The overall goals were generally achieved and in a way such that the simulation can easily be extended with future work. The generic methods and use of open-source maps means that this simulation can be extended to any location or use case with little to no extra work.

7.1 Limits and Recommendations

OpenStreetMap is quite powerful, but as it is written by the community, it can and does have errors. Some simpler errors may be that roads are missing attributes or have the wrong attribute than they have in real life. Another more problematic issue for traffic modeling is not having all of the roads connected or having roads that do not exist. I have encountered both of these issues. Other issues may arise from the fact that the amount of features that a road has (speed limit, name, traffic lights, etc.) depend on the person/people who added/edited the road. Since these attributes are not mandatory to add, relying on them for a simulation is not ideal as there is no automatic way of finding out which roads have the same attributes as they do in real life.

There are some inherent flaws with using OSM maps that cannot be simply fixed like the accidental ones. One major issue is the use of extremely short roads / road segments. For example, a real-life roundabout may be a circle, but OSM will represent it as a polygon with as many sides as there are entrance/exit roads. This can create problems, such as more traffic jams than what is expected, as very short roads will naturally have a very small maximum queue size (as defined by this simulation).

Another issue is that some roads with central medians may be represented as two separate one way roads whereas roads without any median will be a single two way road. This could cause some confusion as to whether a road is truly a one way or is just being represented as two separate roads.

Additionally, if we want to use information from the map such as a stop sign or stop lights, we must still manually check each intersection as there is no guarantee that a signal is correctly placed as some streets may be done more ”lazily” than others as well as some streets may be outdated.

The creation of agents for a simulation is very complex. Beyond the initial complexity of defining realistic agents, we must also validate the agent’s routes through hard to get surveys. These surveys rarely get more than a single digit percentage of the population thus making them not perfect in capturing the travel behavior of the whole population. Additionally, with today’s quickly evolving technologies, surveys conducted may be less relevant. For example, the 2010s introduced private ride-hailing (Uber, Lyft, ...) to the masses, with the latter part of the decade seeing a huge interest in rental electric scooters. These changing modes, along with changing behaviors, means that generating accurate agents remains a complex task.

The simulation itself is somewhat limited in that it relies on external data in order to be perfectly validated. While the results that it produces look correct, further data is needed to validate it especially when considering hypothetical scenarios such as road closures.

For future projects that rely on OSM, depending on the size of the project, more attention can be put into manually checking the OSM map data. If the area of interest is small then manually checking a few dozen roads and intersections is realistic, if the area is large, however, then another approach must be taken such as correcting / checking values automatically through code.

While creation of agents is done in a generic way such that it can easily be replicated for the same or different locations, future work should be completed in this area focusing on how to make agents more realistic for the area of focus, and if possible, generically as well.

Some improvements to the simulation would be improving it through more accurate road conditions such as stop signs, parking, and number of lanes. Weather is also an important factor to the speed of roads, especially when considering other modes of transport such as bicycling.

References

4 agent based modeling examples. (2019). https://mosimtec.com/agent-based-modeling-examples/

Badstuber, N. (2019). London congestion charge: What worked, what didn't, what next. https://theconversation.com/london-congestion-charge-what-worked-what-didnt-what-next-92478

Bazghandi, A. (2012). Techniques, advantages and problems of agent based modeling for traffic simulation.

Bonabeau, E. (2002). Agent-based modeling: Methods and techniques for simulating human systems. *Proceedings of the National Academy of Sciences*, *99*(suppl 3), 7280–7287. https://doi.org/10.1073/pnas.082080899

Easley, D., & Kleinberg, J. (2010). Chapter 2 graphs. *Networks, crowds and markets: Reasoning about a highly connected world* (pp. 23–46). Cambridge University Press.

Hamilton, T., & Wichman, C. J. (2015). Bicycle infrastructure and traffic congestion: Evidence from dcs capital bikeshare. *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.2649978

Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, *4*(2), 100–107. https://doi.org/10.1109/TSSC.1968.300136

Karima, B., Ellagoune, S., Seridi, H., & Akdag, H. (2012). Agent-based modeling for traffic simulation, 51–56.

Kimura, T., Sano, T., Hayashida, K., Takeichi, N., Minegishi, Y., Yoshida, Y., & Watanabe, H. (2018). Representing crowds using a multi-agent model - development of the simtread pedestrian simulation system. *Japan Architectural Review*, *2*. https://doi.org/10.1002/2475-8876.12073

Maidstone, R. (2012). Discrete event simulation, system dynamics and agent based simulation: Discussion and comparison, 1–6.

OpenStreetMap contributors. (2017). Planet dump retrieved from https://planet.osm.org.

Pursula, M. (1999). Simulation of traffic systems - an overview. *Journal of Geographic Information and Decision Analysis*, *3*(1), 1–8.

Rodrigue, J.-P. (2020). 8.4 – urban transport challenges. *The geography of transport systems (5th ed.)* Routledge.

Sisson, P. (2017). 10 u.s. cities getting public transportation right. https://archive.curbed.com/2017/1/24/14361030/best-cities-public-transportation-light-rail-bus

Soriguera, F., Casado, V., & Jiménez Meroño, E. (2018). A simulation model for public bike-sharing systems. *Transportation Research Procedia*, *33*, 139–146. https://doi.org/10.1016/j.trpro.2018.10.086

Viegas, J. (2010). Generating the universe of urban trips from a mobility survey sample with minimum recourse to behavioural assumptions.

Ziemke, D., Metzler, S., & Nagel, K. (2017). Modeling bicycle traffic in an agent-based transport simulation [8th International Conference on Ambient Systems, Networks and Technologies, ANT-2017 and the 7th International Conference on Sustainable Energy Information Technology, SEIT 2017, 16-19 May 2017, Madeira, Portugal]. *Procedia Computer Science*, *109*, 923–928. https://doi.org/https://doi.org/10.1016/j.procs.2017.05.424

Appendix A

NOS Data

In addition to creating a generic simulation, I attempted to extract and build agents from real world data. The data for this additional project was provided by the Portuguese telecommunications company NOS. This data is a record of where and when people connected to the NOS network move between different regions of the Lisbon metropolitan region. The data is anonymous as it only records movements between each defined zone for each hour. The data also only lists the number of movements so tracking an individual person for two or more hours should, in theory, be impossible. The data set lists, for each hour ("hora") of each day of a month, a start ("A") and destination ("B") location as well as the number of people who moved between these locations for that specific hour ("value") (the flux or movement of people). A few lines of this data can be seen in Table A.1. The day is not listed here, but the rest of the mentioned values are. In addition to these, we can see that there is another column "dists". This column is the distance in meters from location "A" (the start or source or src) to location "B" (the destination or dst). While each location refers to a small polygon, the centroid of this region has a defined latitude and longitude. With these values we can calculate distances, assuming 111km to each degree of latitude and 88km to each degree of longitude (a rough calculation for the longitude in which Lisbon lies).

Table A.1: Example NOS Data

hora	A	B	value	dists
1	110105001	110105003	8	2418.7302221279233
7	110105001	110105003	9	2418.7302221279233
8	110105001	110105003	12	2418.7302221279233
9	110105001	110105003	15	2418.7302221279233

As the NOS data already has origins and destinations, the simplest way to use the data is to give each agent a single destination for the whole day. Instead of having an agent go from home to work then back home (as in a full tour), we can instead have 2 agents perform this task (each with a single trip). So one agent is assigned the home to work route and another is given the work to home route.

The benefits of running a simulation in this manner is that we do not need to make any assumptions about how to define what is a ”home” or ”work” location. Assigning agents with data such as the NOS data in, almost makes it ”plug-and-play”. A major drawback to this method with the NOS data is the inability to perfectly sample from the data across all time and location points.

For example, the NOS data contains data for about 3 million origin/destination rows each day in the city of Lisbon. Each of the rows also has the row ”value”, which is effectively a ”multiplier”. Summing all of the values gives around 32 million trips that were technically taken. However, if we are only counting trips taken with cars, we know that there should only be around 600 thousand cars moving around in the city each day.

Therefore we cannot select all possible paths in a day, but must sample. Before sampling, we can begin by eliminating many of the trips as some of the trips have the same origin and destination (people were in the same location from hour to hour), while others are outside of our area of interest (or perhaps time-of-day of interest). Ideally when sampling the data, if a journey into the city is selected, a corresponding journey out should also be. While it is possible to ensure that both of those journeys are taken by agents, since it is not the same agent taking the trip some of the information is lost.

Unfortunately just using the origin/destination matrices as-is is only applicable when we want to look at macro information, not when we want to extract information from individual agents. This section explains the workflow to extract a whole day’s worth of destinations for a single agent (that can then be scaled up for N agents). Note: This method can only be applied when there is data for every single hour of the day (or at least data between the desired starting and ending hours). If not, the section is more applicable.

As we are interested in creating generic weekday O/D pairs for the agents, we must first combine the rows in which the hour, source, and destination are the same. While the each day already has each row with a unique combinations of these, different days of the month will have the same combination appear. This is an important task as we would want to select all of our paths from the same day in order to keep any external events from causing problems. For example, adverse weather should affect all or none of the created paths and not only some of them.

Thus, there are two ways to deal with this problem. The simplest is to simply select a day and filter out all the rows that correspond to other days. The other is to somehow average the flux of people over multiple days. Both methods have their pros and cons, but presumably averaging would produce more average and thus better final results. Choosing one specific day could be useful to see how paths arise from a specific event, weather conditions, etc. from that day. For the following example, a single day, October 10th 2019, was chosen instead of averaging the whole month. No particular reason was chosen other than it is a weekday, which gives us a better representation of commuter traffic than a weekend.

After filtering the data, the next step is to go from discrete and unconnected links between each location for each hour to a connected chain of links that start and finish at the same location (assuming people start and finish their day at a fixed home location) and either move or stay in place for each hour of the day. For example, let there be 2 locations (A and B) for hour 1 and 2. Thus we would get 8 unconnected links (with random fluxes):

Table A.2: Example Links

id	Path	Hour	Flux
1	$A \to A$	1	1
2	$A \to B$	1	6
3	$B \to A$	1	7
4	$B \to B$	1	1
5	$A \to A$	2	6
6	$A \to B$	2	9
7	$B \to A$	2	10
8	$B \to B$	2	6

Now, if we choose location A as the initial start (and by extension final destination), we are left with only two paths. By choosing A, all the paths that start at B for hour 1 are automatically eliminated, as are the ones that end at B at hour 2. We are left with the following 4 possible links to connect:

id	Path	Hour	Flux
1	$A \to A$	1	1
2	$A \to B$	1	6
5	$A \to A$	2	6
7	$B \to A$	2	10

Now, it can be seen that each link has one and only one partner link. This is because we need each link to have a separate hour as well as have the linking destination be the same. So, (A $\to$ B) + (A $\to$ A) is not possible as B $\neq$ A. As it can be seen here, the actual values of the flux (or any other attribute like distance, etc.) is not used in creating a path. This is because we are interested

in first gathering all possible paths and then cutting out the ones we deem unrealistic and not useful. While not using these attributes initially may seem counter-intuitive, we must remember that if we were to define ”realism” now, we would be introducing our own bias. Another reason will be explained shortly.

While the previous example made it seem quite easy to gather the possible paths, the real data is significantly larger. While the example only looked at 2 hours, a real day would be around 17 hours (assuming someone is still at home at 5hr and then returns by 22hr). On the scale of a city, we will also have hundreds or even a few thousand locations. So at each hour we may have several hundred possible links where we need to enforce A = A. Then we would need to enforce this for 17 hours in a row. As we can see that this is not possible by hand, we can implement the links as a graph. Since we will be using a graph, the graph nomenclature of edge and vertex will be used. Edges are the ”links” between each time and location with the vertices the specific time and location.

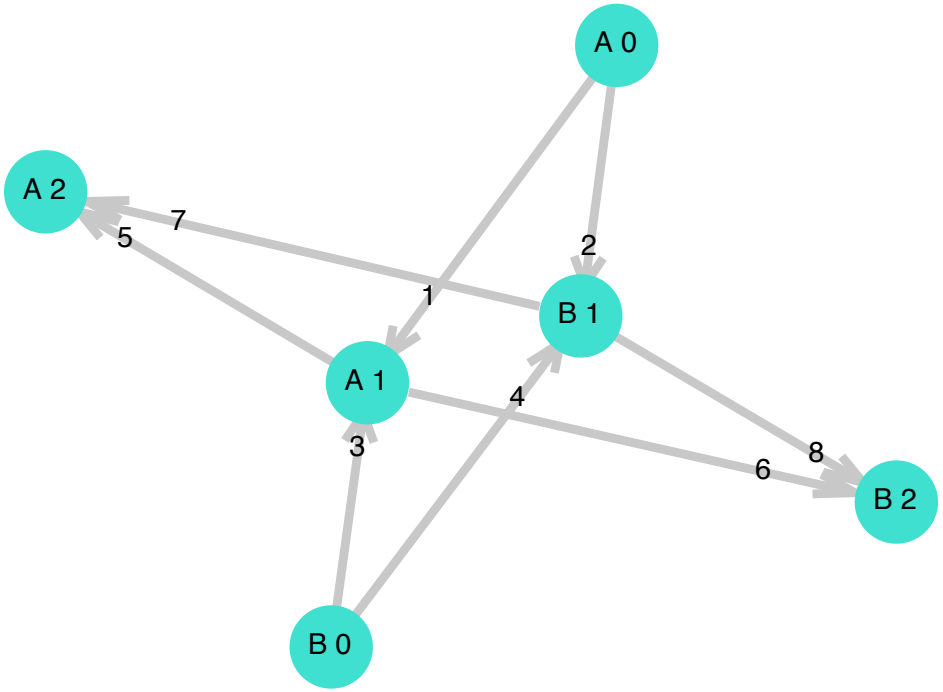


Figure A.1: Graph of Table A.2

In Figure A.1, the numbers on the edges are the ids as given by Table A.2 (and not the flux), and the numbers on each vertex are the hour. Before, we listed each link between two vertices as having a single hour, but in reality the source is in hour k and the destination is in hour k+1. So instead of eliminating the links that are not possible, we can also follow the graph from our start (A 0) to the destination (A 2). As it is a directed graph, we can see that there are only two possible paths (the same as before). Blindly following a link to see if it ends up at the correct destination,

is equivalent to a deep search of the whole space. For each hour we add to our example (while keeping just two destinations), we are looking at the number of possible locations where $N =$ the number of hours. Generalizing this for M locations, M will become impossibly large. 10^{17} is on the scale of the age of the universe in seconds.

As a deep search is impossible, we can make use of one of the many search algorithms available. The A^* is a good choice as it is guaranteed to find the quickest path between two vertices, while also trying to compute it as quickly as possible (so it will certainly be computationally possible unlike the full deep search). The other reason that we did not previously cut out any paths (for lack of "realism") is that the A^* will take care of the idea of realism. It does this by minimizing some cost function, which we may choose to mean maximizing the sum of the fluxes along each individual edge. This also leaves the possibility of creating paths that we may at first glance think of as unrealistic.

Typically, the A^* is quite easy to implement if we know how to define the edge weights. In this case, the weights are not so straightforward. The only attribute that we have for each edge is the flux of people. As A^* tries to find the lowest sum of weights, using the flux as the weight would work. This is because we want the paths with the most flux (amount of people) to be chosen. This would not be a problem if we wanted to find all of the paths possible, but due to computational constraints, it is not possible to find all paths (even though A^* will only return valid paths and the search space is $<< 10^{17}$). Because of this, the best method is to take the inverse of the flux as the weight. Larger fluxes have a smaller inverse and therefore smaller (and more desirable) weight. As the graph does not contain edges with 0 flux, $1 \div 0$ will not occur.

Once the weights are set, the next problem appears. As A^* only finds the shortest path and no others, the algorithm will need to be run multiple times in order to get multiple paths. However, if the weights do not change, the algorithm will always return the same path. So the best way to fix this is to lower the flux weight by 1 (thus increasing the inverse: $\frac{1}{n} \rightarrow \frac{1}{n-1}$ for each edge that is in the previously returned path. This will mean that for the next iteration of A^*, that previous path will now have a larger sum of weights ($1 \div 1$ for each hour). While the same path may be returned again, repeating the change in weight will mean that eventually a new path is chosen. In practice this works quite well with very few paths chosen more than once. The algorithm can then run repeatedly until either a fixed number of paths has been collected or a path that has already be chosen is picked again and its returned weight is 17 (this means no more unique paths are available and that each edge was already at a weight of $1 \div 1$). Once all of the paths have been collected, the process of defining a realistic path and then eliminating the ones that are not, can begin.

The basic outline of the path generation algorithm is as follows:

The first step is to organize and ready the data for ingest into the algorithm. Table A.3 shows the NOS data and how it should look for this particular implementation. The basic features of the input data are the hour, source, destination, and the value (flux or number of people moving between the source and destination for that hour). Each row should have a unique combination of the hour, source, and destination. While two of the three can (and will) repeat, all three should not. The source and destination can list the same location as this indicates the number of people staying in the same location for that hour. Another consideration is the physical location of each source and destination. Before creating the paths, undesirable locations (and every row they appear on) should be removed. Undesirable, for instance, could mean that the location is "out of bounds" of the simulation area. However, since simply removing all locations outside the simulation area will also remove any paths that move in/out of the simulation area during the day, special care should be taken such that the intended behavior occurs.

Table A.3: Path Input Example

Hour	Source	Destination	Value
22	110602002	110626001	7
12	110602002	110626013	7
9	110602002	110635008	7
...	...	...	...

After preprocessing the data, the algorithm begins by running through every single row and counting how many hour and location pairs that exist. This means that the difference between source and destination are not considered at this point. The way that this value is calculated is by creating a dictionary with the hour and location pair being the key. If a key does not exist, then the number of pairs is increased by one and the pair is added to the dictionary. This also gives each pair a unique id.

The next step is creating the graph. The graph is created with as many vertices as there were pairs (last step). Running through each row of the input data (Table A.3), a one-way edge is created between (hour, source) and (hour + 1, destination). Since each row is unique, each edge will be created only once. After the edge is created the weight of that edge is set to the inverse of the value of that row. This means that a higher value (more people) returns a lower weight (which A^* "prefers").

After the graph and weight matrix are created, the actual path finding begins. For every general location found (irrespective of hour or source or destination) in the data, the following loop is performed:

1. Given a starting and finishing hour, the loop will exit if the vertices for the starting or finishing hours do not ”exist”. Some locations may not ”exist” for all hours, such as office buildings, but as long as the location ”exists” at the starting and finishing hour then it will proceed.
2. If they do exist, the algorithm will run the A^* to find the best path between the starting and finishing hours. After finding and recording this path, the weights for each of the edges on that path are lowered by 1 (thus increasing the inverse weight appropriately). The A^* is then repeated endlessly until one of two stop conditions is reached.
3. The loop may exit if it has either reached a defined maximum number of paths or whenever a generated path returns with a maximum weight. This is calculated by seeing if all of the weights in the returned path were already at 1. For such a large graph as this, the former method of loop exiting is almost certainly more likely than the latter.

After each loop for each location is finished, all of the returned paths are saved as separate files (this is to allow for parallel processing). These files can then be combined into a single table for further processing.

Unfortunately, the original data only gives us a small amount of information from which to draw conclusions. This is, of course, intentional in order to keep the original data anonymous and untraceable. What we can deduce, however, is the number of different locations the path visits (as well as unique locations), the distance between each location in the path (and thus the whole distance traveled in a day), and the flux of people from one hour to the next.

With the aforementioned attributes, we can go through each path and make a determination on its ”realism”. One issue is some correlation between these attributes. For example, the number of locations and the distance traveled are naturally linked. Another issue is that by using an average value for a whole day, we lose extreme values for each hour that may be realistic and give us extra information (such as a long drive to and from work, but many hours not moving at work).

The true value of the flux between each location and hour is the fundamental value that we have used for our weights in the calculation of these paths. Unfortunately, this value may be biasing the results that we receive. The intent of using these weights was to eliminate the need to find all possible paths, while still finding realistic and popular paths. The fundamental problem of linking

together two separate paths this way is that it favors higher values of the flux. This means that locations that are highly trafficked will be visited more often (which makes sense), but this also means that paths will choose to potentially revisit these places more often.

We can use the number of locations that are visited along each path to determine whether a path is realistic or perhaps is suffering from an issue of the path bouncing back and forth between two locations. What this means is that instead of a path choosing to remain in the same location from one hour to the next ($A \rightarrow A \rightarrow A \rightarrow A \rightarrow A$) it may follow a pattern like: ($A \rightarrow B \rightarrow A \rightarrow B \rightarrow A$). While there are certainly cases where this is a realistic pattern (delivery / public transportation drivers, pet walkers, etc.), most people (during a standard work day), will not exhibit this sort of pattern. Instead we may expect that a person will remain in the same location for many hours.

The distance between each of the locations in a day (with not moving equal to 0 distance) is quite interesting to look at because it gives us an intuitive way to decide if those sort of movements are possible. For example, if a path has many different locations in a day, but its average distance is quite low, then we may decide that path is more likely than a path with a high average distance. However, the definition of ”high” is up to us. A delivery driver in a truck may drive many kilometers all over the city, so their average distance will be higher than a delivery driver on a bicycle, yet both are still realistic.

Using the time of day own its own is not very powerful. This is because time alone cannot say whether a particular movement between two locations is realistic or not. We have to look at other factors, such as the aforementioned attributes, in order to gauge a path’s realism. Even more powerfully, we can couple the time with the land use of each location (following subsection).

In chapter 4, the OpenStreetMap landuse was discussed and defined. For quick reference, some information will be repeated here as well as the same landuse figure (Figure A.2). The land use is what a particular piece of land is being used for. For example, housing would have a ”residential” landuse tag, schools would have a ”schools” tag, and shops would have a ”commercial” tag.

What we can then do with this information is map it to the locations for each of the origin/destinations. If the location in a path is not probable for a certain time of day or a specific location type is visited more than a realistic number of times, then that path is able to be cut out.

Figure A.2: OSM Land Use (OpenStreetMap contributors, 2017)

After all of the different land use zones are collected, a list of the land use zone names as well as each of their polygon's vertices' ids is collected.

As the vertex ids are stored separately, a second search of the OSM data looks for the vertices found in step 1. When a vertex is found, the latitude and longitude replaces the vertex id in the aforementioned list.

Finally, the most complicated part of the search can begin. First, all of the NOS zones are collected into an array with the NOS id and latitude and longitude of the centroid of that NOS id (the NOS id and the OSM id are not the same).

A loop runs through each of the NOS zones with a second inner loop of all the OSM zones running for each NOS zone in order to find which zone matches that particular NOS zone. Since the ids and zones are not connected by id, we have to take the NOS zone's centroid and check for each OSM zone whether the centroid is inside that OSM zone.

1. If it is inside, then we have found the land use for that NOS zone.
2. If it is not, then we either calculate the closest OSM zone (by latitude/longitude distance) or search around the centroid by changing the angle and distance until a zone is found.
3. However, neither of these methods is perfect as they are guaranteed only to give each zone a land use label and not to give the zone the correct one.

Once all of the NOS zones' land uses have been found, each of the NOS ids, in each generated path, is replaced with its corresponding land use. For a sample region, 47 unique tags were found, and while some tags are quite similar (ones related to various types of green space usage) they are still distinct.

Now that we have an idea of how the algorithm works, we can take a look at a full example using the real data. For any generation of paths, three initial parameters must be set: the region, the number of hours, and the maximum number of paths generated for each starting location.

For this example, the region of Lisbon Figure A.3 was used. This region defines all of the origins and destinations for each hour, rather than just the initial origin / final destination. This means that all paths generated will never leave the region at any point of the day. Allowing agents to start inside a specific zone and go to a different during the day may be desirable, so depending on the desired behavior of the agents, this can be changed correspondingly. However, one consideration is the time taken to find the paths. Using a larger zone will not only take more time as it introduces more starting points, it will also allow for more connections between zones during the day itself (a trade-off between complexity and computational time).

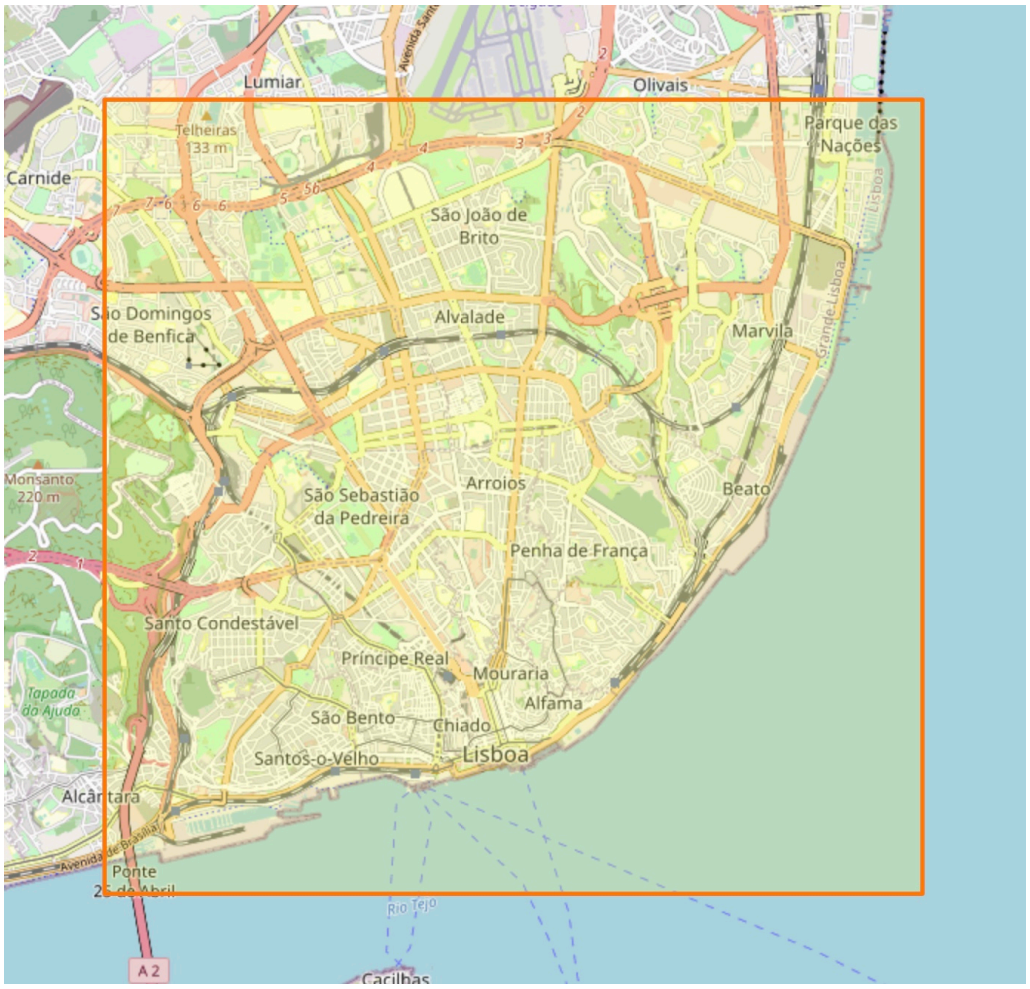


Figure A.3: Lisbon Paths Example

The number of hours chosen for the path is important as increasing the number of hours increases the search space significantly. This is a good thing as it means that more unique paths will be explored and returned. Searching an entire day is also important as we can then see secondary activities (anything besides the main destination (work, school, etc.) of the day). However, a larger search space (longer day) means that it will take more computer time to find every possible path.

For this example, the hours were from 5 to 22. This means that every path has 18 vertices with 1 edge between them for a total of 17 edges.

Defining a maximum number of unique paths that should be returned from the search is important. The main reason was mentioned previously, a large search space takes a long time to find all possible paths. By having an upper limit on the number of paths, increasing the search space with a larger number of hours or number of vertices should not result in a significant increase in computational time (it will have an increase as A^* takes more time with more vertices to search).

A second reason for limiting the number of paths generated is to potentially reduce the number of ”bad” paths that are generated. As the search space includes all possible path combinations, unrealistic paths will also be ”found”. As the paths are chosen in descending order of the sum of fluxes for each origin / destination pair, the first paths will perhaps be more realistic. As more and more paths are generated, paths with lower flux will be returned. This does not mean that these paths did not or could not occur, rather that they were less likely to have occurred.

For this example, the maximum number of paths for each location was set to 5000. Given that the final number of paths is 3.025 million, this means that 605 starting locations were used. Before going through the process of eliminating some of the paths, we can take a look at a few of them on a map just to better help us understand what they look like.

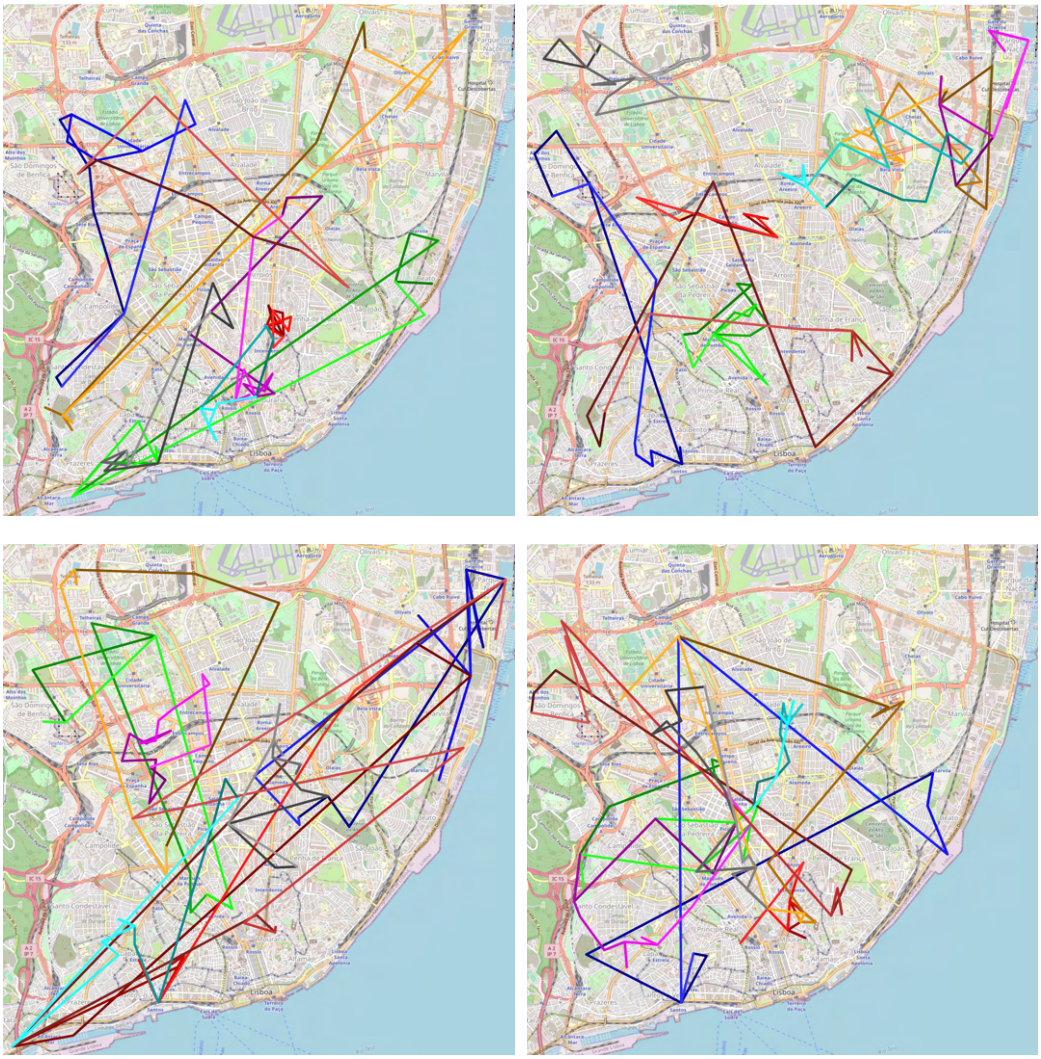


Figure A.4: Example Paths

In Figure A.4, each of the images shows 8 random and unrelated paths. Each of the paths are represented by a single color. The color has a slight gradient, with the darker colors representing the beginning of the path (beginning of the day), and the lighter colors representing the end of the path (day). Even though each of these look realistic enough, we need to reduce the total number of these paths (or sample from them in a smart manner).

After generating paths, we can begin to go through and eliminate paths that we consider to be the most unrealistic. We must do that as according to, the city of Lisbon sees around 600,000 cars per day. However, this takes into account the roughly 220,000 that begin in the city and the 370,000 that enter the city limits from outside. So for the 3 million paths, the target would be 220,000 as the 3 million paths were generated not only from only residences that begin in the city, but with only paths that had their origin and destination in the city. The other 370,000 paths should be generated taking into account locations outside of the city as well (i.e by rerunning the path finding algorithm).

While effort was made to generate these 600,000 agents from the NOS data, in the end it was not successful. While a few paths looked very promising and realistic, the overlap of people's real world paths means that there is no good way to determine if the algorithm has returned a path from one person or several. If an advanced heuristic were used then perhaps more of these paths could be extracted from the data. In addition, the NOS data itself turned out to be somewhat unreliable as it contained several errors in the original data. Even after these errors were removed, the sparsity of the data made it difficult to extract meaningful information from it. The algorithm that I designed for generating agents from the NOS data was not very usable as most of the paths that were returned were simply unrealistic. However, as many paths were seemingly valid, I think that it would still be interesting to explore in the future if the algorithm could be made smarter and/or the original data could be preprocessed better. Some of the problems may also not be fixable as some assumptions, like cellphones are never turned off and the location is perfectly recorded each hour, are almost certainly not true.

While using the NOS data for this project was not successful, it shows a proof of concept for extracting and rebuilding data from an anonymous source data.